

A FLEXIBLE GRAPHICS ADAPTER FOR PARALLEL SYSTEMS

*A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of*
MASTER OF TECHNOLOGY

by
S. GANESAN

to the
**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
MARCH, 1990**

29/3/91
B

CERTIFICATE

This is to certify that the thesis work entitled " A Flexible Graphical Adapter For Parallel Systems " has been carried out by Mr. S. Ganesan under my supervision and the same has not been submitted elsewhere for a degree.

March 1990.

R. N. Biswas

R. N. Biswas

Professor

Dept. of EE

IIT, Kanpur.

SYSTEMS ENGINEERING A

LIBRARY OF THE
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF TORONTO

24 JAN 1991

Th
006.6
G154 f

yd

WATERLOO

CENTRAL LIBRARY
UNIVERSITY OF TORONTO

Acc. No. A.109949

EE-1990-M-GAN-FLE

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF TORONTO
MARCH 1990

ACKNOWLEDGEMENTS

I am greatly indebted to Dr. R. N. Biswas for his able guidance and encouragement throughout the course of this work. The long discussions I had with him were the root cause in bringing this project to a successful completion.

I am extremely grateful to Dr. G. Barua for providing me with valuable reference books, and Dr A. Mahanta for allowing me to use the wire wrap tools. A special thanks to Subramanian, without whom this project would not have been successfully completed. I will always remember him for his help in debugging the hardware. Thanks are also due to the Research Engineers N. R. Gogate and A. Roy for their help in various ways.

I also thank the Image Processing students, Hariprasad and Ramprasad for providing the test image files. Thanks to Ravindra, Kasi and Sanjay for making my stay in I.I.T. a memorable experience. I will always remember the noisy H-mid friends for their cheerful company.

Finally I would like to thank all those whom I wanted to thank but forgot to do so in this page.

CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Graphics and Parallel Processing	2
1.2 Colour and Monochrome Display	2
1.2.1 Intensity Levels	3
1.2.2 Intensity Differences	4
1.3 Reslution and Reconfigurability	5
1.4 Basic Design Objectives	6
1.5 Organisation of Thesis	6

CHAPTER 2 GRAPHICS SYSTEM - OVERVIEW AND DESIGN ISSUES

2.1 Graphics System Configuration	8
2.1.1 Display Memory Contention	11
2.1.1 Display Memory Contention	12
2.2 Layered Graphics Processing	13
2.3 Parallel Processing Graphics Systems	14
2.3.1 Fine-grain and Coarse-grain Parallelism	15
2.4 Advanced Graphics Controllers	16
2.1.1 HD 63484 ACRTC	17
2.1.1 Graphics Coprocessor 82786	20
2.5 Requirements of Flexible Graphics Adapter	23
2.5.1 Portability in Hardware and Software	23

2.5.2 Integrating Text and Graphics	24
2.5.3 Applications to Low-end Systems	24
2.5.4 Configurable Bpp and Resolution	25
2.5.5 High Dot Clock Rate	25
2.5.6 Memory Contention	25
2.5.7 Basic Drawing Operations	25
2.5.8 Scrolling and Zooming	26
2.5.9 Windows	26
2.6 Block Level Overview	26

CHAPTER 3 VIDEO OUTPUT MODULE

3.1 Bit-plane Organization of FB	29
3.2 PS for Bit-plane Storage	31
3.3 Packed-pixel Configuration of FB	32
3.4 PS for Packed-pixel Storage	32
3.5 PS Design	34
3.5.1 Reconfiguration of PS	34
3.5.2 Configuration for Different Bpp	37
3.5.3 Timings for Different Bpp	41
3.6 Synchronization Control	43
3.7 Zoom Control Circuit	45
3.8 PS Control Circuit	47
3.9 Video Generator	50

CHAPTER 4 VIDEO UPDATE MODULE

4.1 Relevant Features of VME Bus	53
4.1.1 VME Read Cycle	53
4.1.2 VME Write Cycle	55
4.1.3 Bus Arbitration Cycle	55
4.2 Video Controller	56
4.2.1 Address Switching in VC	56
4.2.2 VC Design	57
4.2.3 Simultaneous Scan Update Cycle	59
4.2.4 Update by Interleaved Transfer	61
4.3 Direct Memory Access Controller (DMAC)	62
4.3.1 Bus Requester	62
4.3.2 Page Address Register	62
4.3.3 Strobe Generator	65
4.4 Window Manager	65
4.4.1 Window Comparator	66
4.4.2 Window Address Generator	68
4.4.3 Additional Advantages of Windowing	68
4.5 Bus Interface Unit	70
4.5.1 Address Decoder	70
4.5.2 DTACK Generation Circuit	73
4.5.3 Status Register	74

CHAPTER 5 SUPPORTING SOFTWARE

5.1 Device Drivers	76
5.2 Device Driver Routines	77

5.3	Device Driver Implementation	77
5.3.1	<i>fropen</i> and <i>fclose</i> Routines	77
5.3.2	Replica of FB in System Memory	78
5.3.3	<i>frioctl</i> Routine	79
CHAPTER 6	RESULTS AND CONCLUSIONS	80
6.1	Specific Test Results	80
6.2	Implementation Problems	81
6.3	Suggested Improvements	84
	REFERENCES	85
	APPENDIX 1	86

CHAPTER 1

INTRODUCTION

The question "What is Computer Graphics ?" posed to a diverse audience will evoke a wide variety of responses. For a frolicker it may mean video games, while a technocrat may associate it with histograms, pie-charts etc., and an engineer with CAD. The all-pervasive presence and a wide range of demands has resulted in a fast pace of development of graphics from the character-only display CRT of yesteryears to today's state-of-art where even a personal computer have some built-in graphics capability.

The widespread availability of high-performance computers has led to an increased awareness of the importance of visualization techniques like graphics, resulting in enormous expectations by the end-users. The wide spectrum of application of graphics demands stringent quality and often conflicting requirements, leaving the graphics system designer with an uphill task. Due to cost, among many other factors, a user might like to have a choice between colour and monochrome; between the choices of simple monochrome and gray-shade monochrome, he might further want to exploit fully his limited capacity of memory and resolution to the best possible extent. Today the commercial trend is to design a graphics adapter to each of these diverse requirements leading to a variety of adapters flooding the market.

The motivation of the present work is to evolve a Graphics Adapter with sufficient flexibility and modularity to be adaptable to a wide spectrum of computer system including parallel systems. The considerations which are

relevant for deciding the specific design objective for such a system are discussed below.

1.1 Graphics and Parallel Processing

Graphics, in general, involve processing of large amount of data. For example, in CAD it is often necessary to display and edit complex 3D objects, which can take many minutes even on the fastest of processors because of serial data handling. Long processing times may not be critical for scientific applications, but is a serious problem for the interactive, real-time systems used in computer-assisted design and in real time simulations.

The last decade has seen the emergence of processor arrays and networks as practical high-speed computing engines. Many graphics systems have made use of special-purpose drawing processors which speed up a specific application, but have failed to take advantage of the concept of parallel processing. Since many problems in computer graphics are amenable to parallelism, computation intensive tasks of this type can be best performed in real and reasonable time by exploiting the advances in parallel processing. As it turns out, this happens naturally in most practical situations for the simple reason that any computer system using high-end graphics will almost invariably have a bus-oriented parallel architecture. The niceties between the major classes of parallel processing and the ways to implement those are discussed in detail in Chapter 2.

1.2 Colour and Monochrome Display

Although people associate colour almost implicitly with the word 'graphics', one must pause to realize that the inclusion of colour in an electronic display system adds considerable complication to the system, while providing a lot

of visual improvement, and aesthetic satisfaction. Until the early last decade monochrome systems had been enjoying the advantage of having high resolution, but this restriction has been removed by the advent of high resolution colour CRTs, although at a higher cost. Recent studies have shown that, in situations where other characteristics of the image such as size or shape are important, colour may reduce the ability of the user to differentiate these characteristics and adds clutter to the display [1]. Examples of such situations are robot vision, satellite imagery, biomedical applications. Also, because of the lower efficiencies of some colour phosphors, CRTs having these phosphors tend to have lower levels of luminance than monochromatic units and visibility problems arises in well lit surroundings.

1.2.1 Intensity Levels

Displays in continuous use are those at which users spend most of their working days. Systems like this should incorporate such features as larger dot-matrix for character on high-resolution screen, higher frame refresh rate and sophisticated techniques like gray-scale character representation. Use of gray-scale increases the apparent resolution of a display, an improvement that is especially desirable when using a low-resolution display.

Gray-scale character representation is based on the principle that for a point source (a source of light smaller than the resolution of human eye) brightness and size are interchangeable [2]. John E. Warnock of Xerox Parc experimented with font sizes ranging from 5 to 12 pixels in height and with various numbers of gray-levels [3]. His conclusion was that gray-scale character representation substantially improves the appearance of text, even with relatively few gray-levels. At the 5X7 dot matrix character size frequently used in low resolution display, characters become more legible. The effect is obtained

with surprisingly small number of gray-levels; 4 are barely sufficient, 8 adequate and 16 better.

Gray-scale character representation provides the appearance of resolution. The actual number of pixels painted by the electron beam is unchanged. The improved appearance comes from the way the human visual system functions.

1.2.2 Intensity Differences

While the human visual system has special prowess of discriminating edges, these too, have limits. For example, the eye fails to discriminate small differences in intensity levels between two nearby regions, if transition in intensity from one region to the other is gradual, i.e. there is no line of demarcation between the two areas.

This effect can be employed to reduce the impact of staircasing on raster displays. Staircasing effect is inevitably encountered if one tries to draw a straight line which is slightly tilted from vertical or horizontal. Jagged nature of these lines is caused by high contrast between black and white, a contrast to which human visual system is especially sensitive. Picture degradation caused by aliasing is often considered the Achilles' heel of raster scan display devices and considerable effort has been expended to overcome the jaggy nature of lines and edges on raster displays. In general terms the solution to this is to make transition in intensity from the line to the surrounding area smoother. This is achieved by employing shades. Therefore solution to this problem is naturally limited to displays with a image memory that can handle more than 2 intensities per pixel. Any algorithm, no matter what sort of approach has been adopted, ultimately fixes the gray-scale intensity level of each pixel.

To summarize, the gray shade feature in monochrome monitors is too important an asset to be neglected.

1.3 Resolution and Reconfigurability

Resolution of the screen specifies both the number of pixels per line (X-direction) and number of lines in the monitor (Y-direction). Table 1.1 summarizes the resolutions of the commonly available monitors. The wide range of values both in X and Y directions indicates that a well designed graphics controller unit must be able to adjust itself to any given monitor size or at least to a range of resolutions without any change in hardware.

In monochrome monitors each pixel is represented by one bit in the image memory, commonly known as the frame buffer. While for gray-shade monitor the number of bits required to represent each pixel, in short Bits per pixel or Bpp depends on the required intensity levels. For example, for a intensity level of 4, Bpp of 2 and for 256, Bpp of 8 is required. Thus it is obvious that size of frame buffer will increase depending on the intensity level required. But the frame buffer size has to be fixed, because the addressing capability cannot be fully under software control, and moreover the number of address bits has to be predefined and fixed before the system design is undertaken. But from the user point of view it is very much essential to provide flexible resolution capability. Given that the memory size is fixed if one wants to increase the resolution, then Bpp has to be decreased. This is the tradeoff which the user has to live with.

Table 1.1 Commonly Available Monitor Resolutions
Resolution (H x V) | Commercial name |

640 X 200	CGA
640 X 350	EGA
640 X 480	VGA
720 X 348	HGC
1024 X 768	High resolution
1024 X 1024	High resolution
1024 X 1280	High resolution

1.4 Basic Design Objectives

In the light of the above points a flexible graphics adapter should have the following capabilities :

- ✖ It should optimize the available fixed memory resource to achieve the best out of the monitor.
- ✖ Bpp should be programmable to values 8, 4, 2 or 1 enabling 256, 16 or 4 gray shades or monochrome display.
- ✖ It should be adaptable to different resolutions which are currently available as well as which are likely to emerge in near future.
- ✖ As it is meant for operating in parallel environment, it should have the features which make the best use of the capabilities of such parallel systems.
- ✖ The design should be modular and expandable both in terms of resolution and colour.

1.5 Organization Of Thesis

With this preamble, Chapter 2 analyses the features of the present-day graphics systems and identifies their limitations. The block diagram overview of FGA, developed to overcome some of these limitations, is also given in Chapter 2.

Chapter 3 looks into the implementation aspects of Video Refresh logic, which provides the video data to CRT at a fixed rate for display. It mainly concerns itself with the reconfiguration issue emphasized earlier in Section 1.3. It also serves to shed some light on the required analog interface for CRT display.

Once the hardware to display image data present in the memory has been designed, one has to worry about how to transfer data to this memory. Chapter 4 is addressed towards the update logic with which FB can be replenished with new

data. VME bus interface, the key to multiprocessing capability, is also covered in this chapter.

The basic supporting software required for using the assembled FGA is taken up in Chapter 5, the environment being a VME bus-based 32-bit machine running UNIX version BSD4.2. The program listing is given in Appendix 1. Following the tradition, a brief discussion of the results and conclusion is given in Chapter 6.

CHAPTER 2

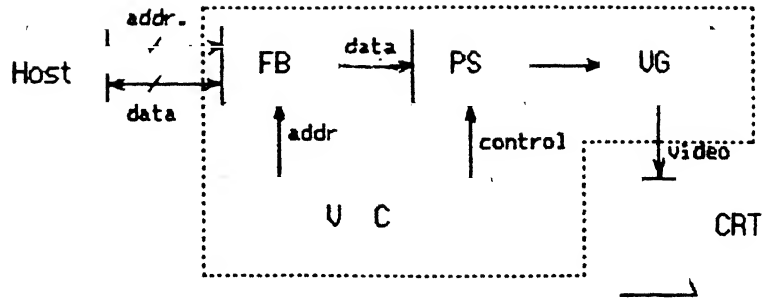
GRAPHICS SYSTEM - OVERVIEW AND DESIGN ISSUES

In this chapter a detailed description of graphics systems as it stands today is presented. Then the viability of doing parallel processing in graphics system is analysed. Graphic processing is broken into different layers and ways to implement these layers is talked about. Functional details of some of the available advanced graphics controller ICs is also looked into. Then a brief discussion on the requirements of and issues involved in the design of *Flexible Graphics Adapter* (FGA) follows. The overview of the designed FGA is given at the end of the chapter.

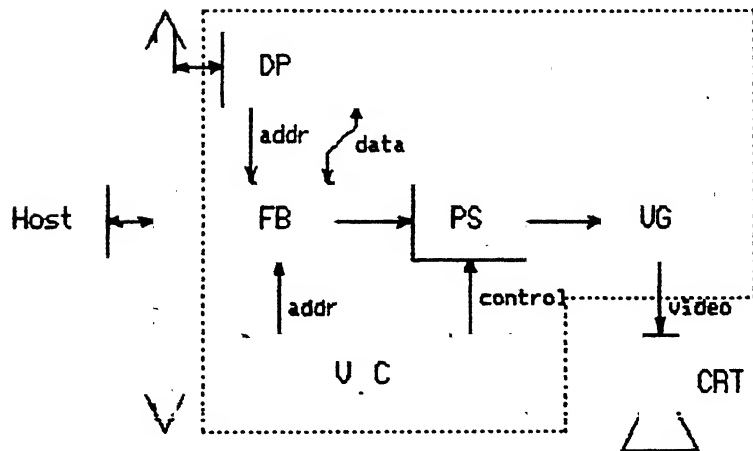
2.1 Graphics System Configuration

The block diagram of a graphics system is shown in Fig 2.1a. The display memory in which the image is stored on a pixel by pixel basis is central to the raster graphics system, and is normally known as the *Frame Buffer* (FB). The Video Controller (VC) generates the address and other control signals to FB during the visible portion of the display. It remains idle during the retrace period allowing FB contents to be read/updated. It has to suitably manipulate the address given to FB, when it has to cater to special functions like pan, zoom, scroll and window.

The data output by FB will be in a parallel form (multiple pixel data packed into bytes) while the CRT requires the pixel data to be given to it one by one. This process of serializing the memory output is carried out by the Pixel



(a) Integral FB Display System



(b) Peripheral FB Display System

FB - Frame Buffer UC - Video Controller
 PS - Pixel Serializer UG - Video Generator
 DP - Display Processor

Fig 2.1 Types of Display System

Serializer (PS), which basically consists of a set of shift registers. This module may be totally absent in graphic systems using Video RAM (VRAM) as VRAMs has an inbuilt shift register. The serialized data is then converted to its analog equivalent which can be directly fed to the CRT monitor.

The mechanism for the loading of data in FB and subsequent updating of FB contents can be of two distinct types – the *Integral frame buffer design* (Fig 2.1a), the *Peripheral frame buffer design* (Fig 2.1b). In the Integral frame buffer design, the update of FB is done by the Host Processor which generates pixel data and dumps into FB which forms an integral part of host memory. In this case, all the processing needed to create a picture has to be done by the host itself. The host gets completely bogged down by the enormity of graphics computation and it is hardly left with any time to perform other general-purpose operations. Hence modern graphics system are not of this type. On the other hand, in Peripheral frame buffer design there is an intermediate Display Processor (DP) which is connected as an I/O device to the host bus and takes commands to update FB. The term *Update access* will be used hereafter to refer to the access by DP/Host, and *Video Refresh access* to refer to VC access.

Most modern high-end graphics systems, therefore, includes a DP which will have the capability for drawing simple figures directly into FB without the intervention of the host processor. DP takes in command from the host via the system bus, interprets them and manipulates the graphics data accordingly. The mundane drawing jobs like line drawing, pattern filling, Bit block transfer operations etc., may be carried out by DP. In cases where the DP can support dynamic RAMs as frame buffer, it has an added responsibility to refresh the DRAM. Memory refresh is done either during the retrace period of the screen or in-between two accesses of the VC.

2.1.1 Display Memory Contention

Update access and video refresh access, both equally important may tend to occur simultaneously, this introduces a situation of potential memory contention. One measure of the performance of a raster graphics system is its level of interactivity, that is, how fast a new image can be generated in response to user input. The entire system architecture affects this figure of merit, but of particular significance is the sharing of FB between update access and video refresh access, significantly constrained by strict video timing standards. For a flicker-free screen the video refresh access should not be disrupted and hence it is given top priority, because of this the update access is restricted and can be performed only when VC is not accessing FB. Dynamic, interactive or real time graphics require that hundreds of pixels be written every frame time, a requirement that demands that update access extend for many memory cycles. VC must have adequate access cycles to the memory so that it can fully refresh the screen.

Advances in technology have aggravated this problem of memory contention. The resolution of the screen has significantly improved, thus increasing the number of video refresh accesses. Processing speeds of processors have increased with the result that pixel data is computed at rates significantly faster than memory cycle times. This results in more frequent update accesses. Cheaper and larger semiconductor memories mean that higher resolution frame buffers can be built with fewer chips but each chip must be accessed more often. Decrease in the memory cycle times over the years is the only factor which comes to designer's rescue.

2.1.2 Reduction of Memory Contention

The condition that update of memory will be allowed only during the frame retrace period to avoid contention, will severely degrade the system's drawing capability, and this may not be a desirable option from user point of view. Since the video refresh access requirements are inflexible, only few alternatives exist for reducing this conflict.

1. Allow VC to obtain more pixels per frame buffer cycle.
2. Provide means to dual-port FB.
3. Interleave the update and video refresh accesses by using fast memories.

The first scheme requires increasing the width of data path between FB and VC. This increase is practical only for widths upto 64 bit, above which component cost and layout consideration tend to make further increase impractical. Therefore a maximum improvement of 4 to 8 fold increase in bandwidth is all that is possible. For further improvement the second scheme which enables concurrent accessing is generally employed. If standard memory has to be dual-ported then special external hardware is required. The more recent graphic systems employ special VRAMs incorporating a large shift register (around 256 bits). When VC accesses the VRAM the shift register is loaded with the pixel data and can then be shifted out independent of other access to memory element. By this method VC demand on FB bandwidth is reduced, and update access can potentially take place at a higher rate. This scheme looks better in all aspects but is an expensive solution because large number of costly VRAMs are required. A slightly better scheme is to use fast memories and interleave the update access and the video refresh access. Most of the advanced graphic controller ICs which will be discussed in section 2.3 have a provision to implement this scheme.

2.2 Layered Graphics Processing

The layered architecture of local area networks is well known. In similar lines graphics processing can also be logically partitioned into 8 distinct layers as shown in Fig 2.2. The figure also shows the operation range of some of important display controller ICs.

The hardware forms the lowest layer and it converts the data sitting in the memory to an appropriate form with which image on the screen can be built. The generation of synchronizing signals needed for the monitor and for overall control of graphics system is by and large done by a CRT Controller (CRTC) like the commonly available MC6845. 6845 also incorporates some low level display control functions like cursor positioning, cursor blinking, interlaced scanning,

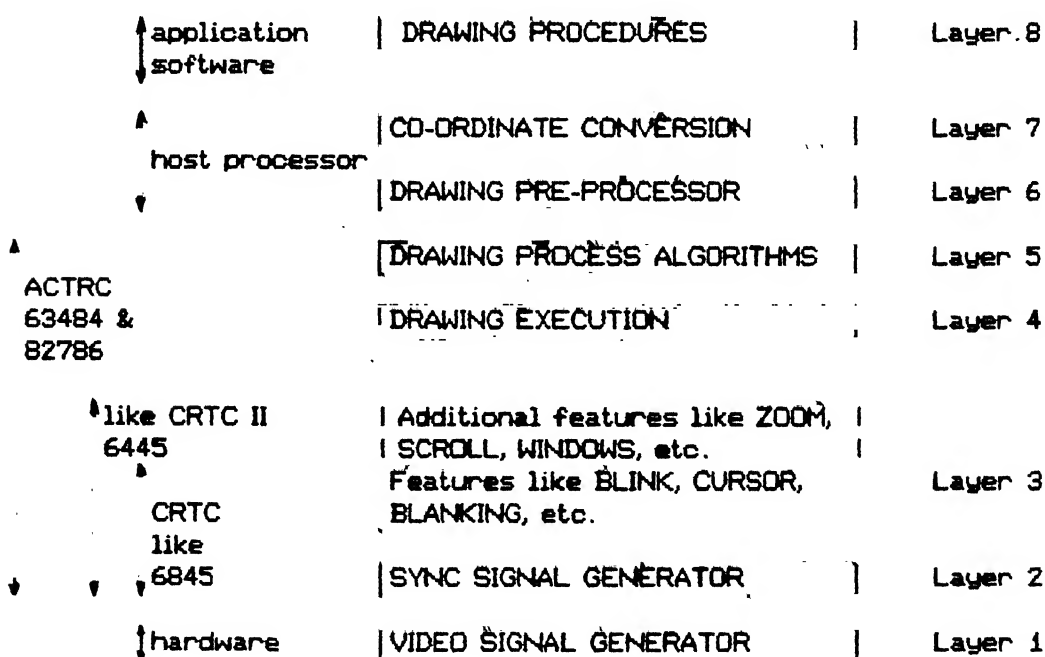


Fig 2.2 Graphics processing layers and ranges of controller IC's

blanking etc. The advanced versions of 6845 are the HD6445 and HD6345, named by the manufacturers as CRTC-II, which in addition to 6845's functions perform some advanced display control functions like smooth scrolling, screen split in horizontal direction resulting in multiple screens and dual cursor.

The drawing process is the execution of particular algorithms to find out the image area to be modified and the new data at these identified locations. The drawing execution is the actual filling of memory with the above computed data at appropriate locations of memory. These two functions are also implemented, in addition to the lower layers, in the advanced graphic controllers like HD63484. Co-ordinate conversion is done basically from object space, where the position is measured in physical units like metres, to image space where position is in terms of normalized screen coordinates. The drawing pre-process includes determining the limiting area beyond which the image has to be clipped. Even in advanced graphics systems the co-ordinate conversion and drawing pre-process are done by the host processor itself. At the top layer are the general-purpose drawing procedures which provide a high-level interface to the user operating system or application software. At this layer, a number of popular standards have emerged including GKS, Core, NAPLP, GSX, VDI and others.

Section 2.4 gives better insight into the two advanced graphic chips HD63484 and 82786.

2.3 Parallel Processing Graphics Systems

As already pointed out in Chapter 1, parallelism improves the performance of graphics system enormously. Peripheral frame buffer design favours well in the speed at which the computation are performed, as compared to the Integral frame buffer design. As an extension to this concept of dedicating a DP to graphics computation, one can go further and realise a much faster system

using multiple DPs. It will be more useful putting general-purpose processors rather than dedicated DPs and realize a parallel processing system. Before going into the design of proposed flexible graphics adapter in such a parallel processing environment, it is in order to compare the important types of parallelisms available.

2.3.1 Fine-grain and Coarse-grain Parallelism

Fine-grain parallel computers consist of a large number (thousands) of simple Processing Elements (PEs) connected by an Interconnection Network (IN). Due to the quantity of PEs, the model of computation is usually SIMD. A central controller broadcasts a stream of instructions to all the PEs to control both the PE operations and IN. Fine-grain PEs are relatively simple, each containing a few registers and a one-bit ALU. Each PE carries out a small fraction of the data-parallel algorithm. Image rendering algorithms fall into this category as they work on a 2D array of picture elements (pixels). Bit block transfer (Bitblt), a primitive operation on a 2D array of bits, can be efficiently performed using fine-grain systems. Referring back to Fig 2.2, fine-grain parallelism means implementation of parallelism at the layers 4 and 5. Update in this case is done by many PEs, while the drawing command comes from a central host processor.

Algorithms utilized in the top layer need much longer range support in terms of data which are often unpredictable. The SIMD scheme will be very inefficient in this case, since data dependent conditional branching is often required. These algorithms, which operate on the display data-structures, can be well executed in a computer system which has coarse-grain MIMD parallelism. By parallelizing the upper layer graphics algorithms in this way one can expect fast response from graphics terminals. Such systems implement parallelism at layers 4, 5, 6 and 7, while the application software gets partitioned into parallel segments

to be executed by different coarse-grain processors.

Software implementors of graphical application systems hesitate to interface their applications to GKS (Graphics Kernel Systems) standard because of resulting system overhead and decrease in performance of simple processor systems. If the system has coarse-grain parallelism the GKS can be distributed on a number of parallel processors to improve system efficiency. One such implementation has been done by Felger et. al. [4].

The above discussion clearly establishes how multiple processor approaches achieve their primary objective, viz enhanced system performance and throughput. A well planned multiple processor system will allow new processors to be added to the system in modular fashion. The other key benefit which accrues from this approach is improved system reliability.

Many systems have been developed in the recent past which exploit fine-grain parallelism while Coarse-grain parallelism has largely been neglected. In this thesis an attempt has been made to recognize the features required for a Flexible Graphics Adapter (FGA) for it to operate in Coarse-grain parallel systems. To decide upon the various features required for the proposed FGA a look into, what the advanced graphic controller ICs can offer, will be helpful to a large extent.

2.4 Advanced Graphics Controllers

In section 2.1 the display processor (DP) and video controller (VC) have been depicted as separate blocks in the graphics system. In the past few years highly advanced graphic controllers have hit the market, in these DP and VC have been integrated to form a single chip. But the controllers are also separately available in IC form e.g. the Am8150 display controller, Am8158 video timing controller and Am8157 video shift register. These form the graphics chip set

released by Advanced Micro Devices. These chips do not possess many of the capabilities of Hitachi's Advanced CRT Controllers (ACRTC) HD63484 or of intel's graphic coprocessor 82786 of intel.

2.4.1 HD63484 ACRTC

The ACRTC meets the functional requirements of powerful visual interfaces, by providing capabilities closely related to those of high level graphic packages. Existing CRTCs provide a single bus interface to the frame buffer, which must be shared with the host. However, the refresh of large frame buffers and the requirement to access the frame buffer for drawing operations can quickly saturate this shared bus bandwidth. The ACRTC uses separate host and frame buffer bus interfaces (Fig 2.3). This allows the ACRTC full access to the frame buffer for display refresh, DRAM refresh and drawing operations, while minimizing the ACRTC's usage of the system bus. A related benefit is that a large frame buffer (2M byte) is usable even if the host has a smaller address space or segment size restriction. While both the bus interfaces exploit 16 bit data paths for maximum performance, the ACRTC also offers an 8 bit mode for easy connection to the popular 8 bit bus structures.

The ACRTC has 5 major functional blocks - Host interface, CRT interface, Drawing processor, Display processor and Timing processor. The host interface takes care of the handshake involved during the communication with the main processor. The CRT interface manages the frame buffer address and arbitrates between drawing address and refresh address. The other three are microprogrammed processors. The drawing processor interprets commands and command parameters issued by the host and performs drawing operations on the frame buffer memory. It is also responsible for the conversion of logical X-Y addresses to physical frame buffer address. Display processor provides display

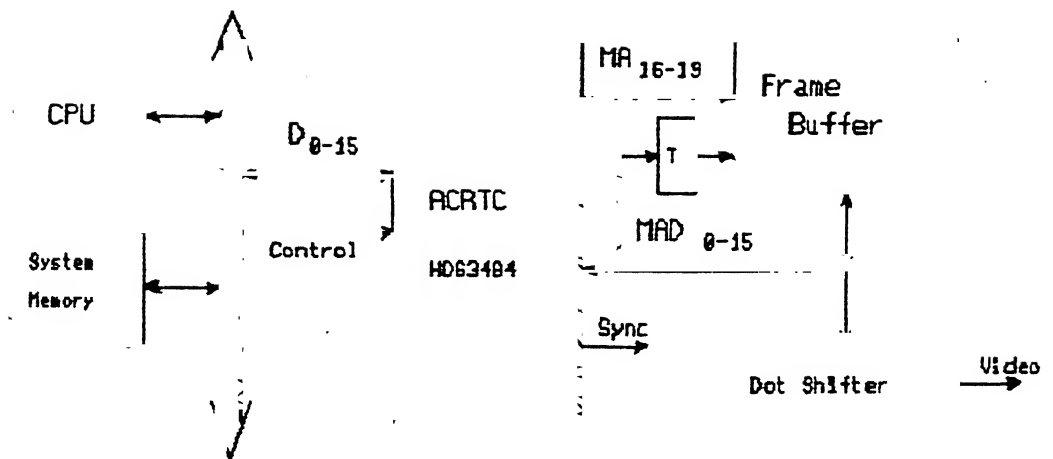


Fig 2.3 HD63484 Based Graphics System

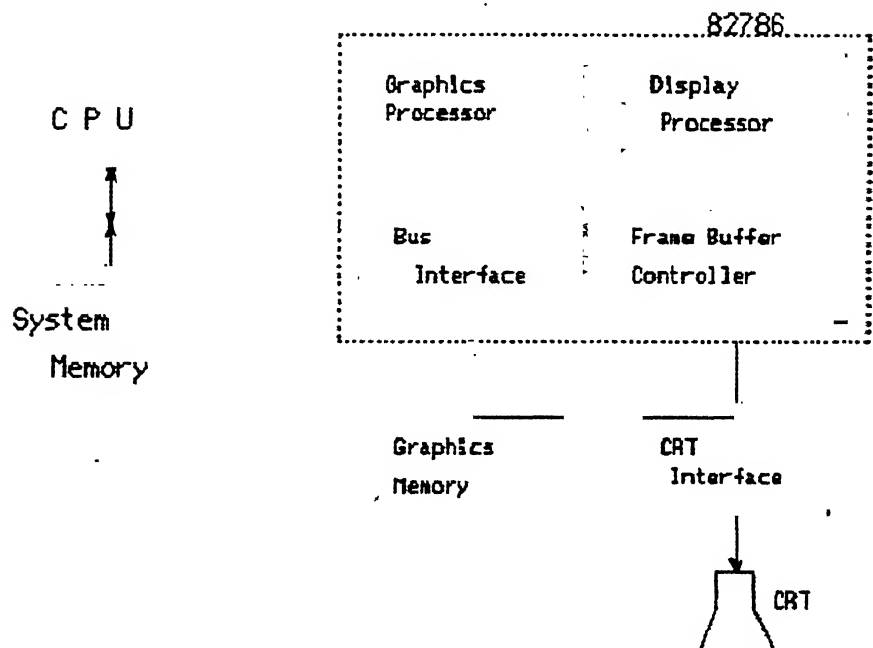


Fig 2.4 82786 Based Graphics System

refresh address. It combines and displays as many as 4 independent screen segments (3 horizontal splits and 1 window).

The timing processor generates the CRT sync signals. The host primarily communicates with the ACRTC's drawing processor via the on-chip FIFOs. The drawing processor takes in 23 different graphic drawing commands for line, rectangle, polyline, polygon, circle, ellipse, arc, ellipse arc, filled rectangle, paint, pattern and copy. It has provision for drawing area control thus providing hardware clipping and hitting. The drawing speed is 2 million logical pixels per second for both color and monochrome.

The display processor can independently do vertical and horizontal smooth scroll for each of the 4 screens. It has the provision for zooming from 1 to 16 magnitude with independent X and Y Zoom factors. While zooming and scrolling the contents of the frame buffer remains unchanged, only the image on the screen changes. The display processor can handle DRAMs as frame buffer by providing them the required refresh address.

The ACRTC allows the division of the frame buffer into 4 separate logical screens – upper, base, lower and window screens. The first three form the background screen group. In the simplest case, only the base screen parameters must be defined. Other screens may be selectively enabled, disabled and blanked under software control. The background screens partition the display into three horizontal splits whose position is fully programmable. A typical application might use the base screen for the bulk of user interaction, using the lower screen for a status line and the upper screen for pull-down menu. The window screen is unique, since the ACRTC gives it higher priority than background screens, which get overlaid. The ACRTC has 3 cursor modes – the normal Block mode, graphic mode (in this the cursor pattern and shape is programmable) and the cross-hair mode.

2.4.2 Graphics Coprocessor 82786

Intel's 82786 (Fig 2.4) actually comprises two concurrent processors—the Graphics processor (GP) and the Display processor (DP) — with high-level instruction sets. GP draws shapes and text into memory to create images. DP retrieves appropriate portions of memory to create the desired display. The CPU has direct access to the graphics memory and is generally used to write commands for GP and DP. However, the CPU can also draw or transfer pictures in graphics memory (This facility is not available in HD63484). 82786 being a coprocessor can access system memory directly.

The frame buffer controller, which is tied to the bus interface, is smart enough to take advantage of special fast-access addressing such as page mode, static column mode and nibble mode which are available in DRAMs. The 82786 uses it not just for video refresh but also for Bitblt operations and filling. As a result, the chip can rapidly fill sequential locations in horizontal areas. The 82786 allows for windows to be created from separate bit maps, each of which can be generated by a separate application and can use a different number of bits to represent each pixel. With the window management of this chip, a programmer can simply alter pointer values to move text or graphics within windows or to move entire windows; there is no need to redraw the contents of a frame buffer each time the display contents are changed.

Fig 2.5 shows the roles and interactions of the CPU, DP, GP in creating and displaying pictures. The CPU generates a command list —resembling a program describing the picture to be drawn — for GP, and places it in graphics memory. As GP executes the command list a picture is formed in a bitmap in the graphics memory. Through this process, the CPU can make GP draw an unlimited number of pictures on the bitmaps in memory. To display the pictures that have been created on screen, the CPU generates a screen descriptor list in graphics memory for DP.

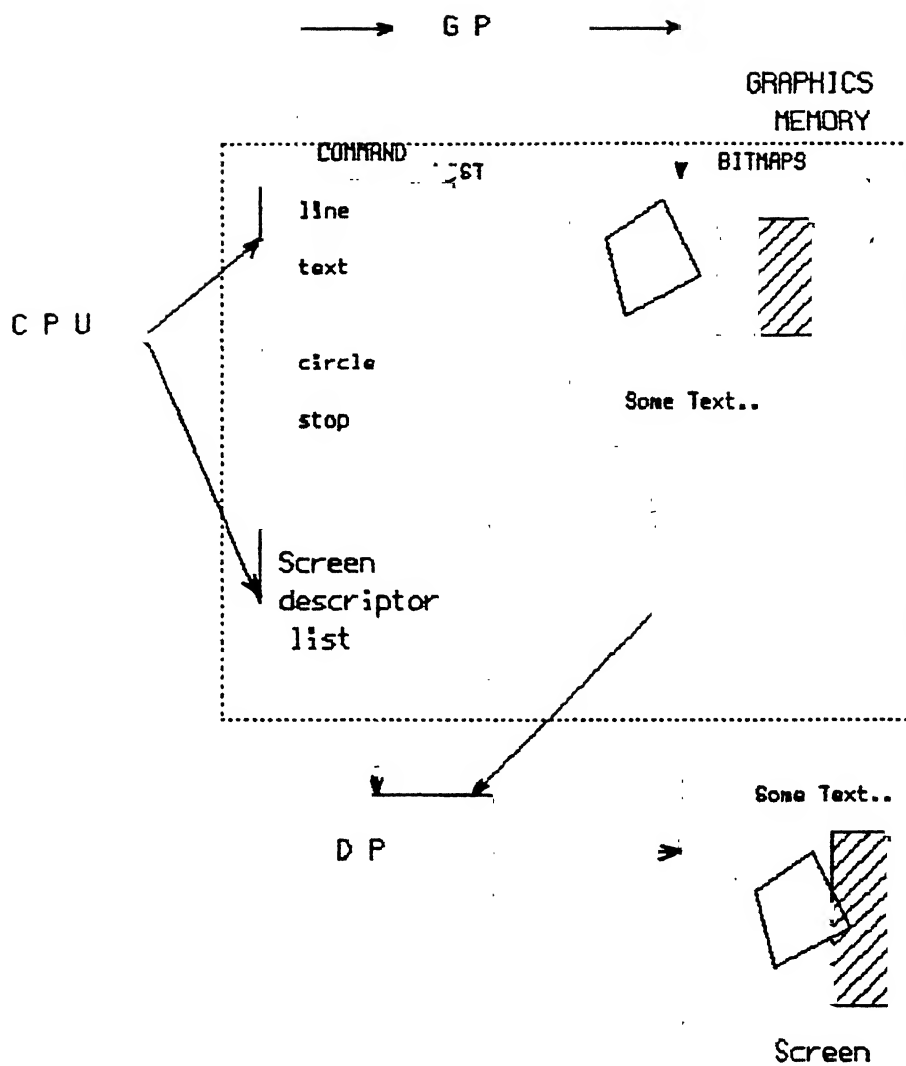
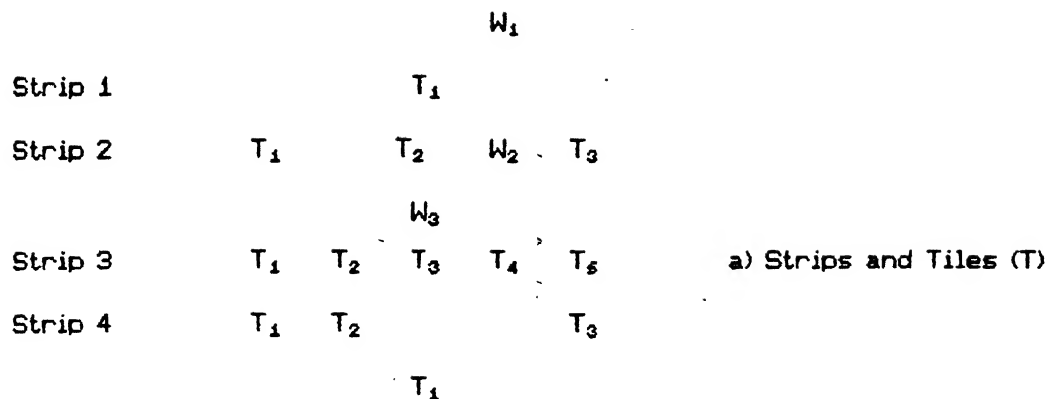


Fig 2.5 82786 Based System Interaction

Using this information DP extracts portions from bitmaps and displays them at specified location on the screen. This creates windows on the screen. Each window can display any portion of any bitmap. GP draws and DP displays while remaining totally transparent to each other. The graphics memory is used to store command lists, screen descriptor lists, character fonts, pictures (bitmaps) etc., and serves as an information exchange medium between the processors.

The bus interface of the 82786 is designed to support all 16 and 32 bit processors from Intel. A special 8 bit mode is also provided as in Hitachi's counterpart. The 82786 can draw all standard primitives – lines, circles, polylines, polygons – like the 63484, thus relieving the CPU from the chore of creating these, in complex figures, from scratch.

The display processor module divides the screen into *Strips and Tiles*. A new strip starts at the border of windows on the Y axis and a new tile begins at a window boundary in X direction (Fig 2.6). Each strip is divided into one or more tiles, dividing the screen into a mosaic of tiles. For a given screen the host provides a screen descriptor list, which is a linked list consisting of a set of strip descriptors specifying the number of scan lines and the number of tiles in each strip, and a set of tile descriptors specifying the width, contents and attributes associated with the objects being displayed. Screens are assembled in real time, without the delay usually involved in reading out the contents of a bit map located in memory and reproducing it on the screen. It does this by reading the strip and tile descriptors and filling an on-chip FIFO cache with the pixel data from the bit maps in the tile descriptors. The FIFO cache is then emptied onto the video screen. The contents of upto 16 tile descriptors can be cached on the chip. As a result, when it is necessary to move a window or windows or to scroll the contents of a window, only the strip and tile descriptors need to be changed; the bit map contents can be left undisturbed.



[Strip 1 header]
 | Tile 1 |

[Strip 2 header]
 | Tile 1 |
 | Tile 2 |
 | Tile 3 |

b) Screen descriptor list

Fig 2.6 Strips and Tiles

2.5 Requirements of the Flexible Graphics Adapter

The purpose of the above section was to identify the features that comprise the present day graphic controllers. The following sub-sections identify those features which are essential so that they can be implemented if possible in a improvised manner in the proposed FGA.

2.5.1 Portability in Hardware and Software

FGA is envisaged to work in parallel processing systems. In such an environment the system design should be modular so that processors can be added, just by plugging in new cards, resulting in an increase in processors

operating in parallel. For this the interface of all units must be done to a standard bus e.g. the VME bus. FGA should also have an appropriate bus interface so that it can work in the relevant bus-compatible systems. The low-level software which directly communicates with the hardware must also be portable. This can be achieved by developing software for FGA operation, in standard operating system environment like UNIX.

2.5.2 Integrating Text and Graphics

Although text is conceptually just another form of graphics data, in practice it is often treated as distinct entity. In recent years memory prices have fallen to the point that is economical to represent both text and graphics using bit maps, though text was earlier stored in ASCII format as it conserved space. Many applications require text and graphics to be displayed simultaneously. Advantages are manifold if text and graphics are both stored in the frame buffer as bit patterns. The font size and shape will become programmable. The space between two adjacent characters can be varied depending on the character width (this is better known as proportional spacing). All graphics facilities like powerful Bitblt will also become available to text.

2.5.3 Application to Low-end Systems

Since FGA is primarily addressed to high-end systems it is required to be interfaced to a standard bus with substantially large data-bus width, which is a maximum of 32 bit in most standard busses. This is necessary because of large amount of data associated with any graphics display system. But sometimes FGA may need to operate in low-end systems having 16 bit data bus width or still lower. Though efficiency and performance of FGA will sharply come down, it should still be able to operate in a such a system providing at least the barely

necessary functions to the user.

2.5.4 Configurable Bpp and Resolution

FGA would not be truly flexible if its Bpp and resolution cannot be programmed by software commands. The available standard adapters like EGA, VGA, CGA should become a subset of FGA. It should also cater to high-resolution graphics for which no standard is available.

2.5.5 High Dot Clock Rate

The dot clock rate is the frequency at which the pixel data is shifted out of PS. With the advancing monitor technology the resolution of the CRT is on a constant rise. Increased horizontal resolution means more pixel data has to be read from memory and shifted out through the shift registers in a scan line. The only way to handle this is to increase the dot clock rate. As FGA should handle high-resolution monitors, it should be able to operate at frequencies far above those used in standard adapters.

2.5.6 Memory Contention

Much has already been said about frame buffer memory contention problem. It will make FGA more efficient if the memory contention can be tackled in a better way than the existing methods.

2.5.7 Basic Drawing Operations

Doing primitive drawing operations in hardware will surely be a desirable feature. But as FGA is meant for working in coarse-grain parallel systems, on-board computing power is redundant. Hence FGA in the present design would not be able to operate on the frame buffer memory contents to change the

displayed image.

2.5.8 Scrolling and Zooming

These two features, when implemented in hardware makes software programming so simple, that these get incorporated in the design more readily. FGA should follow this trend.

2.5.9 Windows

From the discussion of 82786, it is clear that hardware windows give enormous amount of power to application programs. The number of windows in the 63484 is limited to 1, and the 82786 will be bogged down if there are more than 16 tiles or windows per scan line. With the increasing resolution and more complex application programs this limitation becomes a handicap. It is desirable to have a scheme designed with which hardware can handle any number of windows per scan line. Theoretically speaking, the number should be limited only by imagination.

2.6 Block Level Overview

An evolutionary analysis of the architecture of frame buffer display system has been done in the previous sections. The integral frame buffer design (Fig2.1a) is too taxing on the host while in the peripheral frame buffer design (Fig2.1b) the drawing speeds are restricted by the speed of the display processor. In high-end graphics systems this might slow down the graphics processing heavily and DP may prove to be a bottleneck. DP in fact inhibits the exploitation of parallel processing even if it is available as part of the overall system.

The proposed FGA configuration is therefore chosen to be an altogether different display system architecture. The block diagram of FGA along

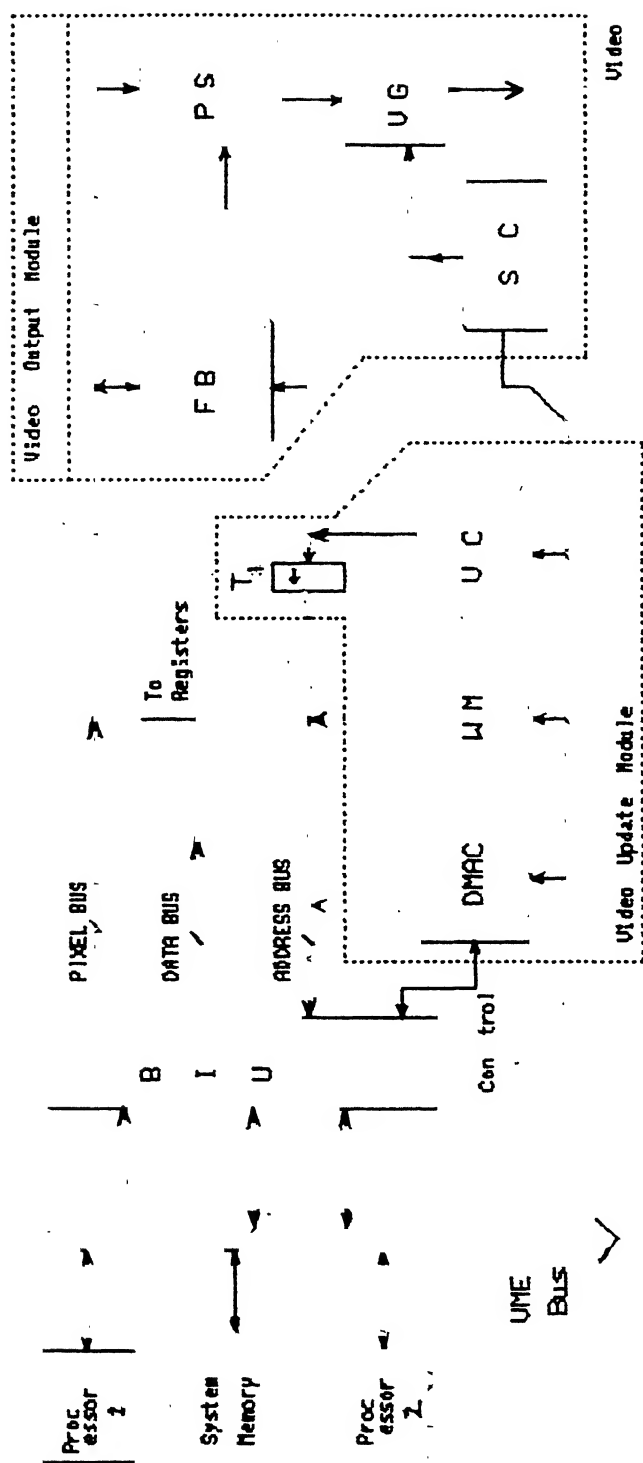


Fig 2.7 Block Diagram Of FGA

with the system bus interface is shown in Fig 2.7. The host system is chosen to be a VME bus based one due to its availability. FGA has a DMA Controller (DMAC) which does the block transfer of screen or window data from system memory. This relieves the processors from doing the job of transferring hundreds of kilobytes of image data. Since there is no intermediate DP in the proposed FGA, the graphics processing can take the advantage of parallel processing fully. The FGA can be functionally divided into two parts — the *Video Update Module* and the *Video Output Module*. Both these modules in FGA are interfaced to VME bus through Bus Interface Unit, which basically consists of bidirectional buffers.

The Video Update Module consists of the DMAC, Window Manager (WM) and Video Controller (VC) blocks. The DMAC does the job of procedural acquisition of the bus, following certain specific protocols. DMAC also provides the address of the page from where the data has to be transferred. WM provides the DMA address when a window and not the background has to be updated. VC performs the usual video controller's job of generating the necessary address sequence for FB.

The Video Output Module consists of the FB, Pixel Serializer (PS), Video Generator (VG) and Synchronization Control (SC) blocks. The output of FB is transferred through the Pixel bus and gets shifted out by PS. The binary bit pattern of pixel intensity is converted to its equivalent analog video signal, amplified and fed to the CRT by VG. The video refresh address has to be suitably controlled when scrolling or zooming is to be done. The necessary control signals and clocks are provided by the Synchronization Control (SC) block.

The design of Video Output Module will be discussed in the next chapter, while details regarding Video Update Module is given in Chapter 4.

CHAPTER 3

VIDEO OUTPUT MODULE

The overall block diagram of the FGA given in Fig 2.7, comprises two functionally independent modules. This chapter is addressed to the design issues for the Video Output Module, which consists of the following blocks – Frame Buffer (FB), Pixel Serializer (PS), Video Generator (VG) and the Synchronization Control (SC). The requirements of PS would depend on the configuration of FB, bit-plane organisation being the most common, though not necessarily the most efficient. In the next two sections, the pros and cons of a bit-plane design are therefore taken up before we finalise our actual FB configuration.

3.1 Bit-plane Organization of FB

The well known bit-plane arrangement of FB stores multiple pixels in a byte of a chip. In Fig 3.1 P_{ij} represents bit j of i th pixel, e.g. if Bpp=8, the i th pixel is represented by 8 bits $P_{i1}, P_{i2}, \dots, P_{i8}$ spread over 8 memory planes and each byte in a memory plane consists of bits representing 8 consecutive pixels. The number of bit-planes varies with the value of Bpp. The name bit-plane comes from the fact that the successive bits in FB correspond to successive pixels of display. Pixel intensity is represented by the corresponding bits in all the bit-planes. So this means that multiple pixels are accessed from a bit-plane every time a read or write is performed.

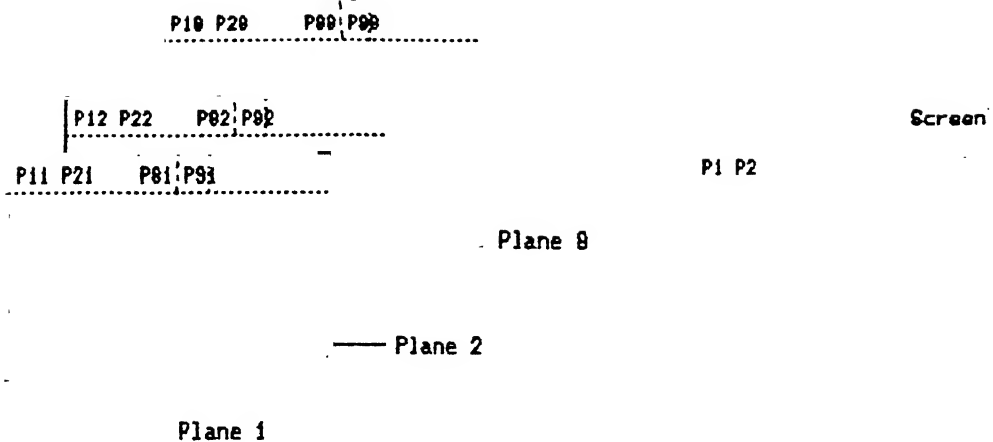


Fig 3.1 Bit-plane Arrangement For $Bpp=8$

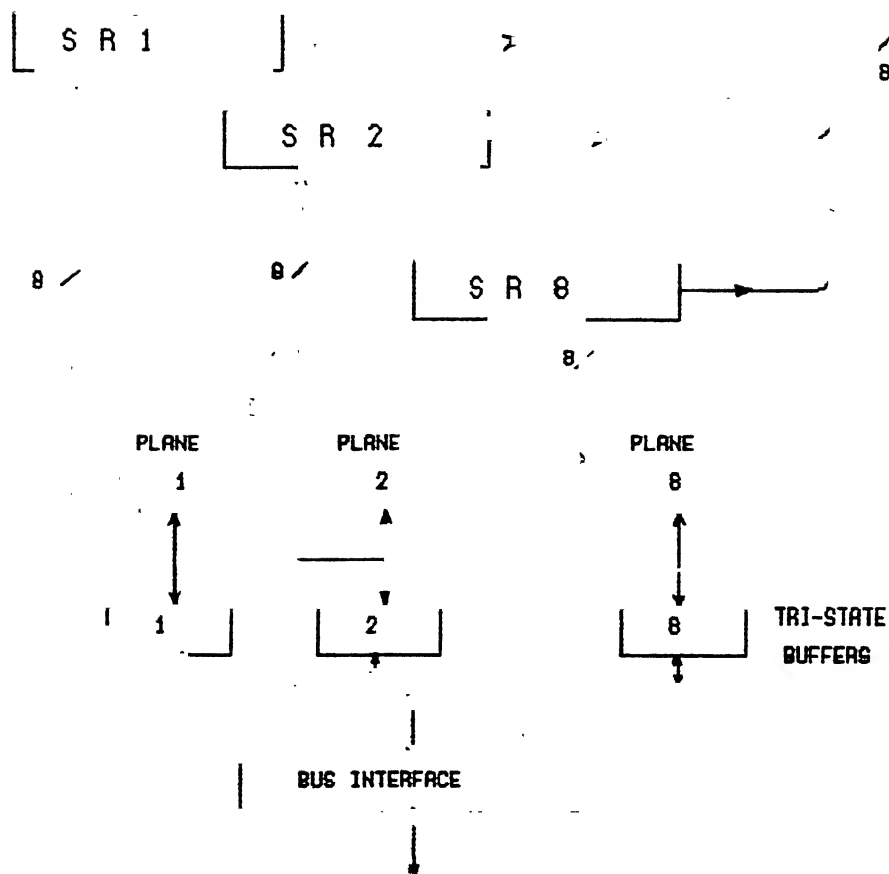


Fig 3.2 PS For Bit-plane Storage

3.2 PS for Bit-plane Storage

Serialization of parallel data is the job of PS and this can be done easily by using shift registers. An arrangement of PS is shown in Fig 3.2, where Bpp=8, and the address to the bit-planes which are simultaneously accessed during a video refresh cycle is provided by the Video Controller (VC). The data output from the 8 bit-planes are parallelly loaded into 8 different Shift Registers (SR) which are of the parallel-in serial-out type. For every serial clock applied to these SRs all the 8 bits representing a particular pixel are parallelly available at the outputs of SRs. These bits are latched and then converted to analog by Digital-to-analog converter (DAC). It should be noted that the data bus lines coming out from the different bit-planes do not have any physical interconnection between them.

Because of the facts that commercially available RAMs have bidirectional data bus and that for an update access the bus width cannot be as large as 64 bits, it is necessary to use tri-state buffers as shown. Using these tri-state buffers different planes can be updated in different cycles, by enabling the buffers cyclically. So to modify a single pixel we need to perform multiple write cycles, depending on the width of the system data bus, e.g. for an 8-bit system, 8 write cycles will be required to modify a pixel. Moreover, when a pixel value has been read in this way, 8 read cycles would result in the accumulation of 64 bits of data. The processor has to extract the required pixel data of 8 bits from this 64 bit data, which when done for each and every pixel might cause a severe software overhead. One advantage with bit-plane systems, which is worth mentioning, is the ability to access a single bit-plane alone. This capability will be of significant help with systems using overlays. To summarize, the disadvantages of the bit-plane memory arrangement are :

- * Additional tristate buffers are required, the number being equal to the

number of planes, resulting in additional control circuitry and cost.

* To read or write a pixel multiple R/W access has to be performed. This could be a severe constraint, e.g. when a vertical line has to be drawn in the screen, as the vertical line will generally encompass one or 2 pixels in horizontal direction.

* Severe software overhead may result because the required pixel has to be extracted from multiple bytes which are accessed.

Because of the above three major disadvantages it is decided not to adopt the bit-plane method of storage for FB. The other alternative arrangement of memory is packed-pixel format described in the next section. This arrangement is supported by major graphics controllers like HD63484 (ACRTC).

3.3 Packed-pixel Configuration of FB

In packed-pixel format there is no concept of planes of memory. Each byte in a memory may hold the intensity information for 1/2/4/8 pixels, accordingly as Bpp = 8/4/2/1. Therefore for each read access at least 1 pixel data is output as shown in Fig 3.3, unlike in the bit-plane arrangement.

3.4 PS for Packed-pixel Storage

In packed-pixel storage format the number of bits to be accessed at one stroke is neither fixed nor related to Bpp, and can be chosen according to the designer's convenience. This is in contrast with the fact that in bit-plane arrangement the number of bit-planes parallelly accessed has to be equal to Bpp. The access times of commonly available static RAMs are around 100 to 150 ns. This means that with pixel times (It is the time in nanoseconds taken by the electron beam to cover the distance between two adjacent pixels) aimed at 40 ns (corresponding to a dot clock rate of 25 MHz), at least 4 pixel data will have to

be accessed simultaneously in order to load the shift registers in time with valid pixel data. As the VME bus interface provides 32 bit data path, one can fix the width of the access to be 32 bit implying that the number of pixel data simultaneously accessed from the system memory will be 4, 8, 16 or 32, accordingly as Bpp = 8, 4, 2 or 1. In this arrangement, tri-state buffers which are discussed in the earlier section would not be required as the data paths always match. As already mentioned, 8 different SRs are required to display an 8-Bpp image, and hence 8 four-bit shift registers are provided in PS as shown in Fig 3.4.

It can be seen that the severity of the disadvantages mentioned for the bit-plane have been reduced to a great extent in this arrangement. If the update bus is 32 bit wide then tri-state buffers are not needed and a single R/W access is enough to modify a pixel. This observation will not hold good for low-end systems having 16 or 8 bit busses. Though in this FB arrangement also, a pixel has to be extracted from 32 bit data, the extraction process merely involves ANDing by 1's at appropriate places in the longword, and is therefore achievable with a single machine language instruction.

3.5 PS Design

The ability of PS to adapt itself, with the help of external commands, to a particular Bpp needs careful analysis and the following sub-sections concentrates on this aspect.

3.5.1 Reconfiguration of PS

Referring back to Fig 3.4, let us denote the stream of bits coming from the first 4-bit SR as S_{11} , S_{12} , S_{13} and S_{14} and similarly for other SRs. Over and again it has been stressed in the previous chapters that flexibility in terms of

Bpp is the key feature of FGA. The eight SRs which are purported to constitute PS obviously do not have this capability. They always shift out 8 bits simultaneously for every shift-clock. This means that the 32 bits loaded are coming out in batches of 8 so that $32 \div 8 = 4$ pixels are formed out of the loaded bits. This is all right for Bpp=8, but for Bpp=4 we need 4 bits to come out in parallel. So the loaded 32 bits have to come in groups of 4 resulting in the formation of $32 \div 4 = 8$ pixels. Similarly for Bpp=1 the stream has to fall on a single line to produce 32 pixels.

A method to achieve this reconfigurability is shown in Fig 3.5. Fig 3.5a is essentially the same as Fig 3.4 with the SRs shown as solid rectangle and the bits constituting the pixels as dotted rectangles. Thus in the figure a box of 8 bits has to be shifted per shift-clock; this can simply be achieved by using 8 SRs which are indicated. These parallelly output 8 bits, then have to go to an 8-bit DAC for analog conversion. In Fig 3.5b, we need to output 8 pixels, and hence we have to do 8 shifts with the available 4 bit SRs. One way to achieve this is to connect the SRs in tandem as depicted in the Figure. Now the rectangular box encloses only 4 bits indicating that 4 bits are shifted out per shift clock. These bits can then be fed to a 4-bit DAC.

For the single Bpp configuration if we try to extend the idea of tandem connection then the data flow sequence will have to be

SR8 → SR7 → ... → SR4 ... → SR1 → Video Output

This requires a totally different connection pattern between serial input and serial output pairs of SRs. Alternatively, a way to shift the bits has been devised using an additional 8-bit shift register (Fig 3.5c). The dotted box shows the respective bits which are loaded simultaneously into the extra 8-bit shift register. The scheme works as follows : The 8 bit output from the 8 SRs gets initially loaded into the extra SR. These 8 bits then, get shifted by 8 shift

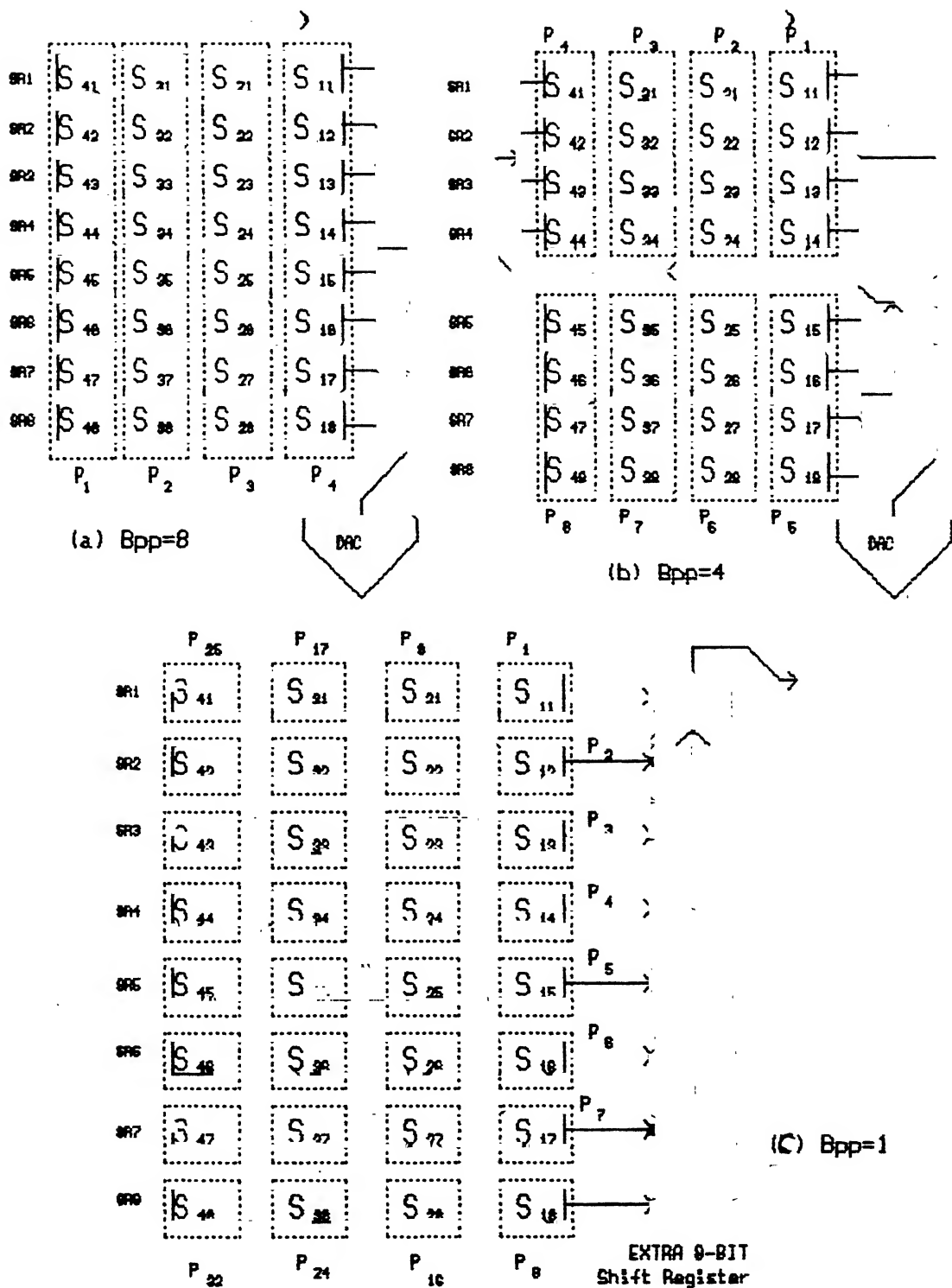


Fig 3.5 Shift Register Configurations For Different Bpp

clocks to appear as a bit stream at the output. After this the next 8 bits from the SRs get parallelly loaded and the process repeats. The bit stream can be directly sent to the monitor after suitably buffering and no DAC is required in this case. In this stream of bits, the location of bit 0 corresponds to S_{11} and this is fixed as the connection is hard-wired. So whatever may be the Bpp configuration the relative positions of bits in the stream are fixed and therefore the packing scheme for other Bpp's cannot be varied to a convenient pattern. With this packing scheme, the consecutive pixels for Bpp=4 occur after every alternate nibble and every fourth pixel occupies consecutive nibble locations. With this pixel pattern for Bpp=4 the FB cannot be, strictly speaking, called as packed-pixel storage. For Bpp=1 the packing is straight-forward and falls exactly into the definition of packed-pixel scheme.

The Fig 3.6 shows the complete implemented circuit of PS. The distribution of FB output bits is not shown in the figure to avoid confusion but the data pins are labelled from D_0 to D_{31} to indicate connectivity. This 32-bit data bus in between FB and SRs is called as the *Pixel Bus*. The shift registers, 74S195, are of 4-bit Parallel-in Parallel-out type. The need for two 8-bit latches will be explained in the following sub-section. These latches are basically shift registers acting as extra shift registers for Bpp=1 and as latches for other two cases. The shift clock given to the SRs are indicated as Hclk and their corresponding load pin as HSH/ \overline{LD} . Similarly the latch cum shift register has Vclk and VSH/ \overline{LD} as the clock and load inputs. To differentiate the clocks of the two 8 bit latches Vclk(up) and Vclk(dn) will be used.

3.5.2 Configuration for Different Bpp

The flow of data through PS for Bpp=8 is shown in Fig 3.7a. The inputs to the SRs are not shown in this figure. The data in the registers gets right-

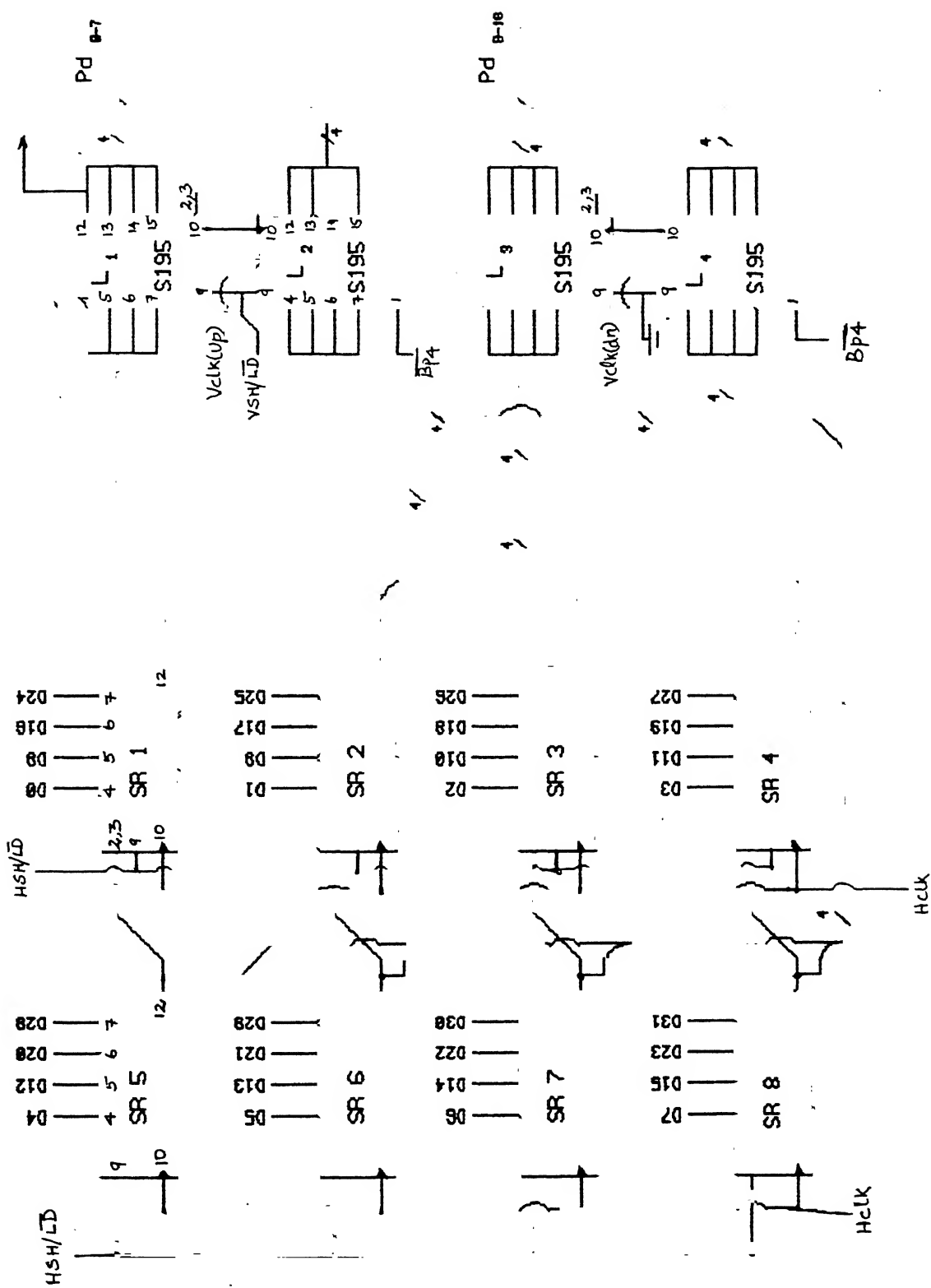
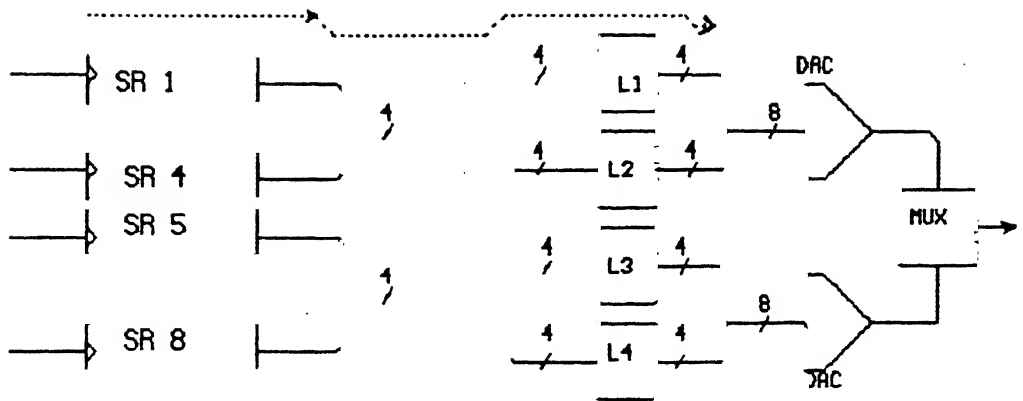
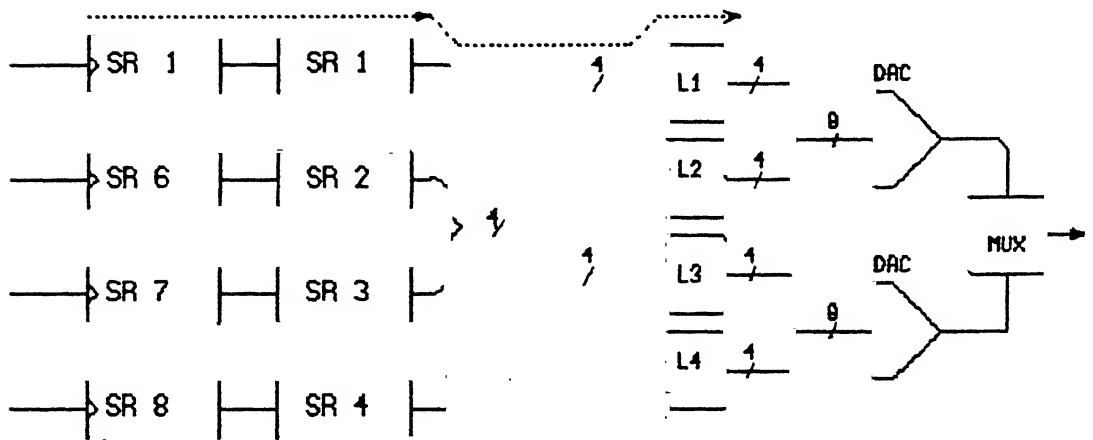


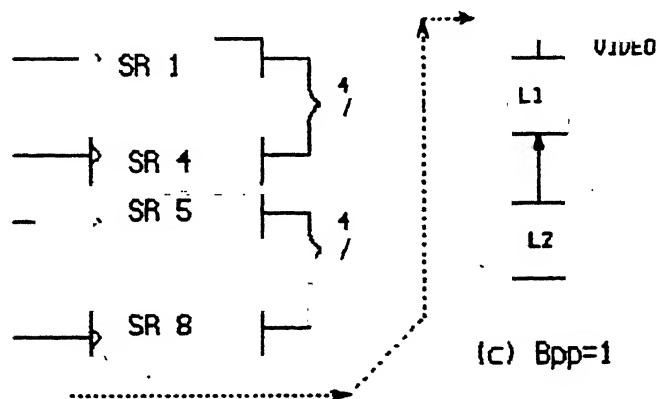
Fig 3.6 Complete Implementation Of PS



(a) Bpp=8 Configuration



(b) Bpp=4 Configuration



(c) Bpp=1 Configuration

Fig 3.7 Different PS Configuration

shifted for every Hclk pulse. The shift registers are synchronously loaded when \overline{LD} (load) pin of S195 is at logic 0. In 8-Bpp configuration the SRs are configured as eight 4-bit shift registers; The 8 bit output from the SR is fed to the latches L_1 to L_4 which get appropriate control inputs to act as a latch and not as a shift register. The serial data output by SRs gets latched alternatively either to L_1 and L_2 or to L_3 and L_4 . The data movement through the SRs and latches is shown by dotted lines.

The need for 2 banks of latches and DAC arises from the need to avoid cumulative delay. This follows from the fact that when one of the pixel is getting written on the screen the other pixel can be converted. To be more precise, when the analog mux output is connected to I_0 , DAC_1 can do the conversion for the whole pixel time, so that when mux output is switched to I_1 the converted value is ready and waiting there.

The reconfiguration for 4-Bpp display is shown in Fig 3.7b. The data input connection of L_2 and L_4 though not shown are still existing but the \overline{CLR} input of these latches are made low so that the lower 4 inputs of DAC are always 0. The 74S195's are configured in this 4-Bpp case to form four 8-bit shift registers. As can be seen from the figure, the SRs are cascaded. The serial output of SR_5 is connected to serial input of SR_1 to effectively form a 8-bit shift register. Similar connections exist between SR_6 and SR_2 ; SR_7 and SR_3 ; SR_8 and SR_4 . The 4 bits simultaneously output by SRs get latched either to L_1 or to L_3 .

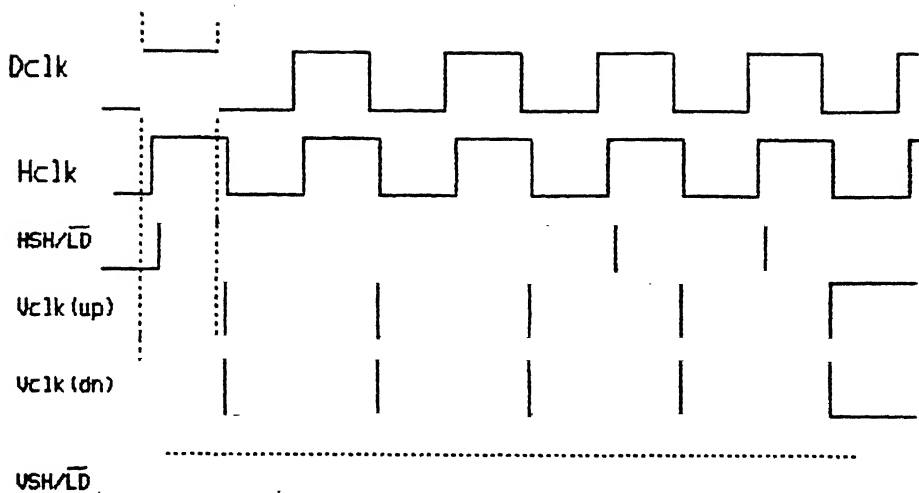
The reconfiguration for the 1-Bpp case is shown in Fig 3.7c. The configuration is same as for 8-Bpp case except that L_1 and L_2 now act as a shift register instead of latch. The data is not loaded into L_3 and L_4 and DACs are not used in the monochrome configuration. For every 8 shifts in L_1 and L_2 there is one shift in SRs. The video bit stream comes through the serial-output of L_1 .

3.5.3 Timings for Different Bpp

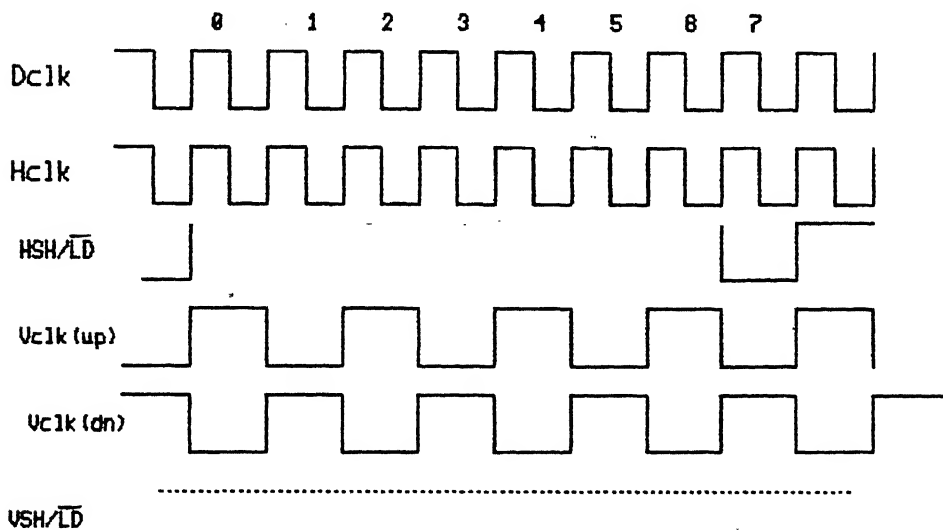
When the display is providing 256 gray levels, the timing analysis diagram is shown in Fig 3.8a. The Dclk in this figure is mnemonic for dotclock, which is the rate at which the Video Generator is outputting pixel data to the CRT. The Hclk is same as Dclk except that it is delayed. The reason for this and other delays in the figure are due to the propagation delay through the control circuit of PS. Volk(up) and Volk(dn) have always 180° phase difference between them. In this case they are generated by dividing Hclk by 2. VSH/ $\overline{\text{LD}}$ is always at logic '0' thus enabling the S195 to act as a latch. The HSH/ $\overline{\text{LD}}$ is basically Hclk divided by 4, thus SR₁ to SR₄ get loaded every fourth Hclk pulse, making them act as 4-bit shift registers.

In 4-Bpp case (Fig 3.8b) HSH/ $\overline{\text{LD}}$ = Hclk ÷ 8. So the shift registers gets loaded once after 8 shifts thus acting as a 8-bit shift registers. The VSH/ $\overline{\text{LD}}$ is low as in 8-Bpp case. All the other conditions remain same as for the 8-Bpp configuration.

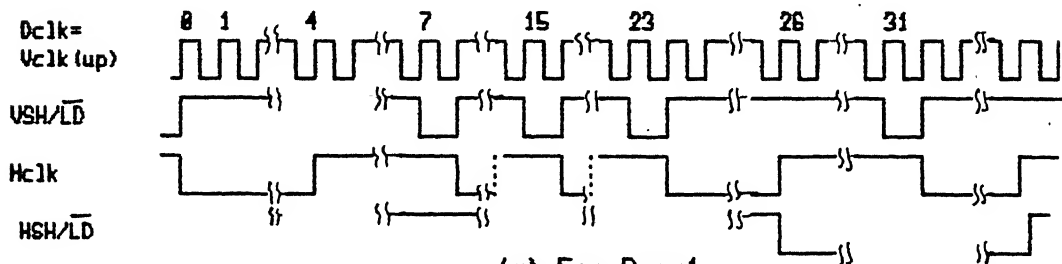
From the timing diagram of 1-Bpp case (Fig 3.8c) it is clear that the horizontal shift registers SR₁ to SR₈ get loaded after 32 Dclk pulses. Now the Volk(up) is same as the Dclk which is the shift rate at which pixel data is sent to CRT. The registers L₂ and L₁ are cascaded by connecting serial output of L₂ to serial input of L₁. So the horizontal shift registers along with the vertical ones (L₂ and L₁) form a 32 bit shift register. The VSH/ $\overline{\text{LD}}$ is now pulsed by the Dclk ÷ 8 signal. Thus the vertical shift registers get loaded after shifting 8 pixel data vertically out. After every vertical load the horizontal shift registers get right-shifted by one position. The Hclk is basically therefore ÷ 8 signal of Volk(up), but since S₁ to S₈ should be shifted only after the VSH/ $\overline{\text{LD}}$, the same signal cannot be given as Hclk. So to get a delayed pulse, the Q₂ output of the counter which divides the Dclk to generate these $\overline{\text{LD}}$ control signals is given as Hclk. After 4



(a) For Bpp=8



(b) For Bpp=4



(c) For Bpp=1

Fig 3.8 Timing Waveform For Different Bpp

such shifts, taking $4 \times 8 = 32$ Dclks the horizontal shift registers get loaded once again. So $HSH/\overline{LD} = Dclk + 32$.

From the above paragraphs it is clear that the Dclk has to be divided by different factors depending on Bpp for which the PS is programmed to operate. Table 3.1 summarizes the different signal frequencies in terms of Dclk for varying Bpp.

3.6 Synchronization Control

Central to any clock generator is the crystal. A 50MHz quartz crystal has been used for the present design. The clock generator circuit is shown (Fig 3.9a). As FGA has to operate in a wide range of frequencies, this 50 MHz is divided by a free-running up-counter which provides clock frequencies of 25, 12.5, 6.25, and 3.125 MHz. The required Dclk can be selected by changing the control bits given to the MUX. These control bits come from the Status Register, which is discussed in the next chapter.

To produce the horizontal synchronization (Hsync) and Vertical synchronization (Vsync) signals, MC6845 (CRTC) is used (Fig 3.9b). The programming of the chip's internal registers is done by using the signals RS, \overline{CS} , R/\overline{W} , E and data bus. The Register Select (RS) input of CRTC is connected to the address bit A1 of VME bus. As the address bus and other special functions like light-pen detection provided by CRTC are not used, the need for reading CRTC never arises

Table 3.1 : Control signal frequencies for different Bpp

Bpp	HCLK	HSH/ \overline{LD}	VCLK(dp)	VSH/ \overline{LD}	VCLK(dn)
8	Dclk	Dclk/4	Dclk/2	0	Dclk/2
4	Dclk	Dclk/8	Dclk/2	0	Dclk/2
1	Dclk/8	Dclk/32	Dclk	Dclk/8	-

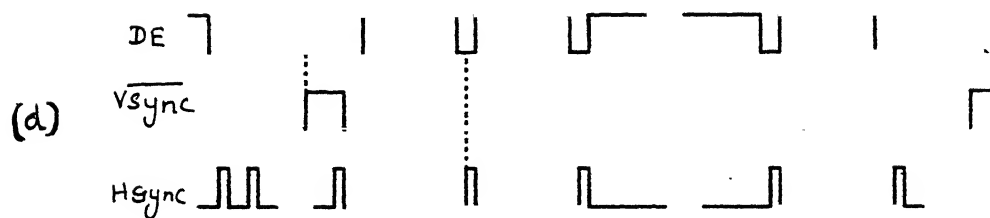
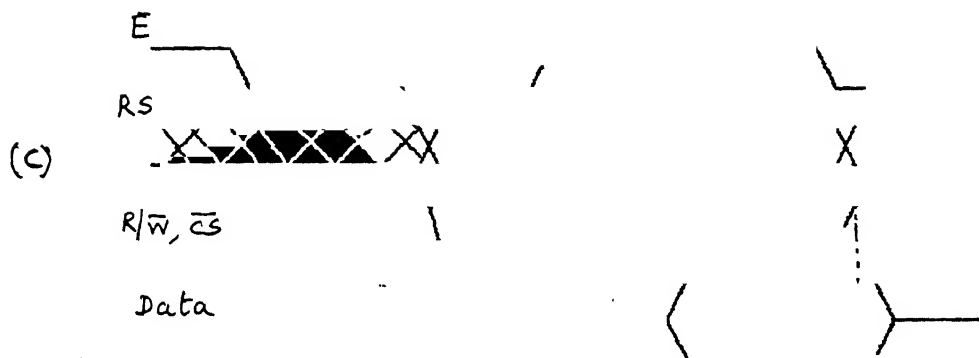
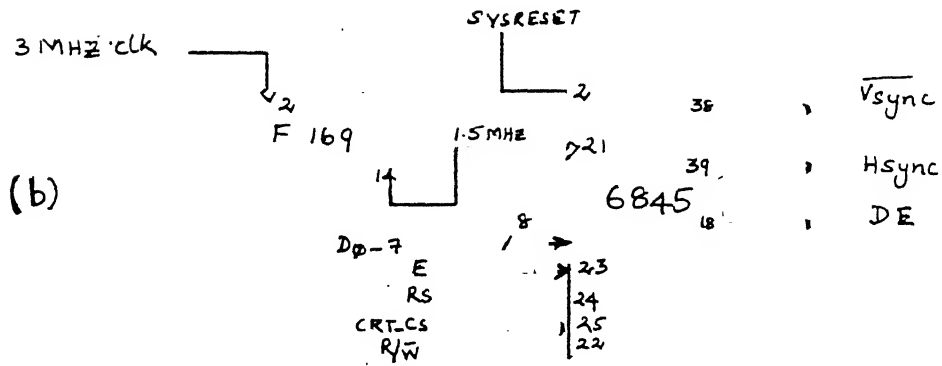
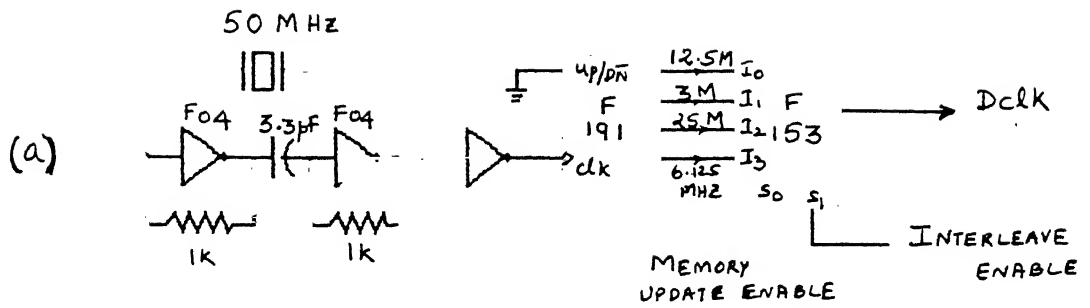


FIG. 3.9. SYNCHRONIZATION CONTROL CIRCUITS

and \bar{CS} can be tied to R/\bar{W} . The timing waveform for initializing 6845 is shown in Fig 3.9c. These waveforms are provided by the Bus Interface Unit, design details of this will be taken up in the next chapter. The character clock which is given to CRTC is generated by dividing the 3.125 MHz clock by 2. The 6845 can operate upto a maximum character clock frequency of 3 MHz. The relationship between the various outputs generated by CRTC -- the Display Enable DE, Hsync, $V\bar{S}\bar{Y}\bar{N}\bar{C}$ are shown in Fig 3.9d.

3.7 Zoom Control Circuit

To achieve zooming, the clock given to the shift registers has to be slowed down. To get a zoom factor of 2 the clock has to be divided by a factor of 2 so that the pixel time seen by the shift register is doubled and essentially same pixel data will be sent to analog output circuitry for 2 DCLKs. In general, zoom is done in vertical and horizontal directions by the same amount, though modern graphic controller IC's like HD63484 provide separate zoom factor for both X and Y directions. To zoom along vertical direction the VC has to repeat the same address for multiple scan lines. To accomplish this DE has to be divided by zoom factor.

The zoom control circuit is shown in Fig 3.10a. It has a zoom factor register, 74LS174, which is a hex D flip-flop. Five of these bits are used -- 4 for zoom factor, 1 bit indicating whether zoom is enabled or not. The I/O address map of this register is 0xffff7c. There are two down-counters -- DCLK divider, DE divider -- both of which are 74F169. The Terminal Count (\bar{TC}) is the signal obtained by dividing the clock input to the counter, by zoomfactor programmed. The DCLK divider is enabled to count only during the visible portion of the raster. The 2:1 MUX allows the Zoomed clock (ZDCLK) and Zoomed DE (ZDE), if zoom enable bit is set, otherwise unzoomed equivalents of these are passed.

D0-5 CS7

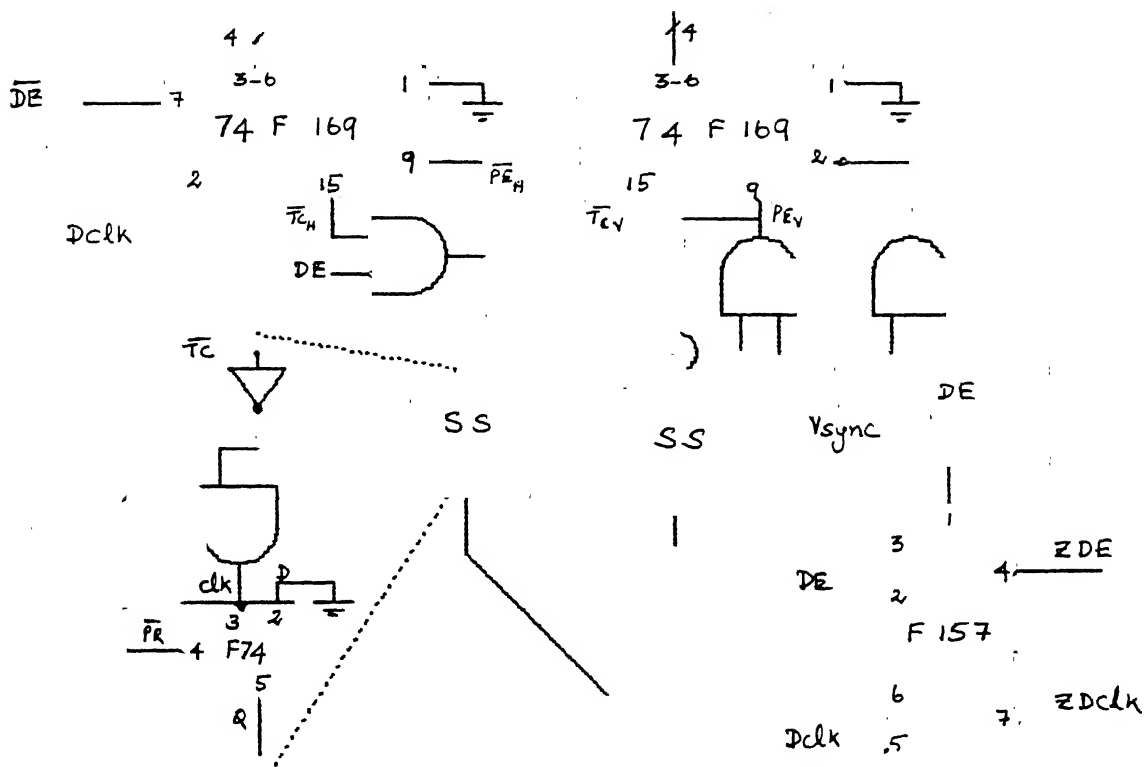
6

74 LS 174

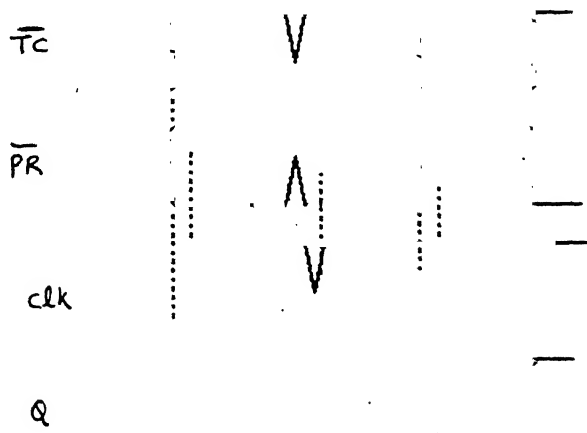
Q0-Q3

Q4

4 / ZOOM ENABLE BIT



(a) CONTROL CIRCUIT



(b) SPIKE SUPPRESSOR WAVEFORMS

Fig 3.10 ZOOM CONTROL CIRCUITRY

Since 74F169 is synchronously presettable, to load these counters clock should be present when \overline{PE} is low. The Dclk divider normally gets loaded on the input clock's rising edge, whenever \overline{TC} is low. This ensures continuous counting. Since one clock is wasted in loading, the factor by which the division has to take place should be one more than programmed value. Thus, to divide by 2 the counter should be loaded with 1. This observation holds good for all circumstances concerning loading of F169 and will not be explicitly mentioned hereafter. The clock divider should be ready with loaded zoom factor before the first clock comes in to decrement it. So the counter has to be loaded during the retrace period, i.e. when DE is low. To achieve this DE has been given as one of the control inputs to an AND gate, so that \overline{PE} is low when DE is low. For the similar reason Vsync controls \overline{PE} in the DE divider. In this the clock also has to be given to initialize the counter, hence Vsync also controls the clock input of DE divider. These two observations that \overline{PE} should be low and clock should be present at appropriate time to initialize the counters, is true for most of the counter circuits to be discussed later.

The \overline{TC} of either counter is not given directly to the MUX input but passed through a flip-flop which acts as Spike Suppressor. The \overline{TC} signal is prone to decoding spikes hence if it is used as clock signal or asynchronous reset signal erroneous operation will result. The relevant waveform are shown in Fig 3.10b and the circuit is self explanatory.

3.8 PS Control Circuit

The major component of control circuit of PS (Fig 3.11) are the counters (74F169) which divide the ZDclk. The various parallel load signals for the shift registers are obtained by dividing the ZDclk by the appropriate factor, which is determined by the contents of the Bpp register. The status register (74LS74) is

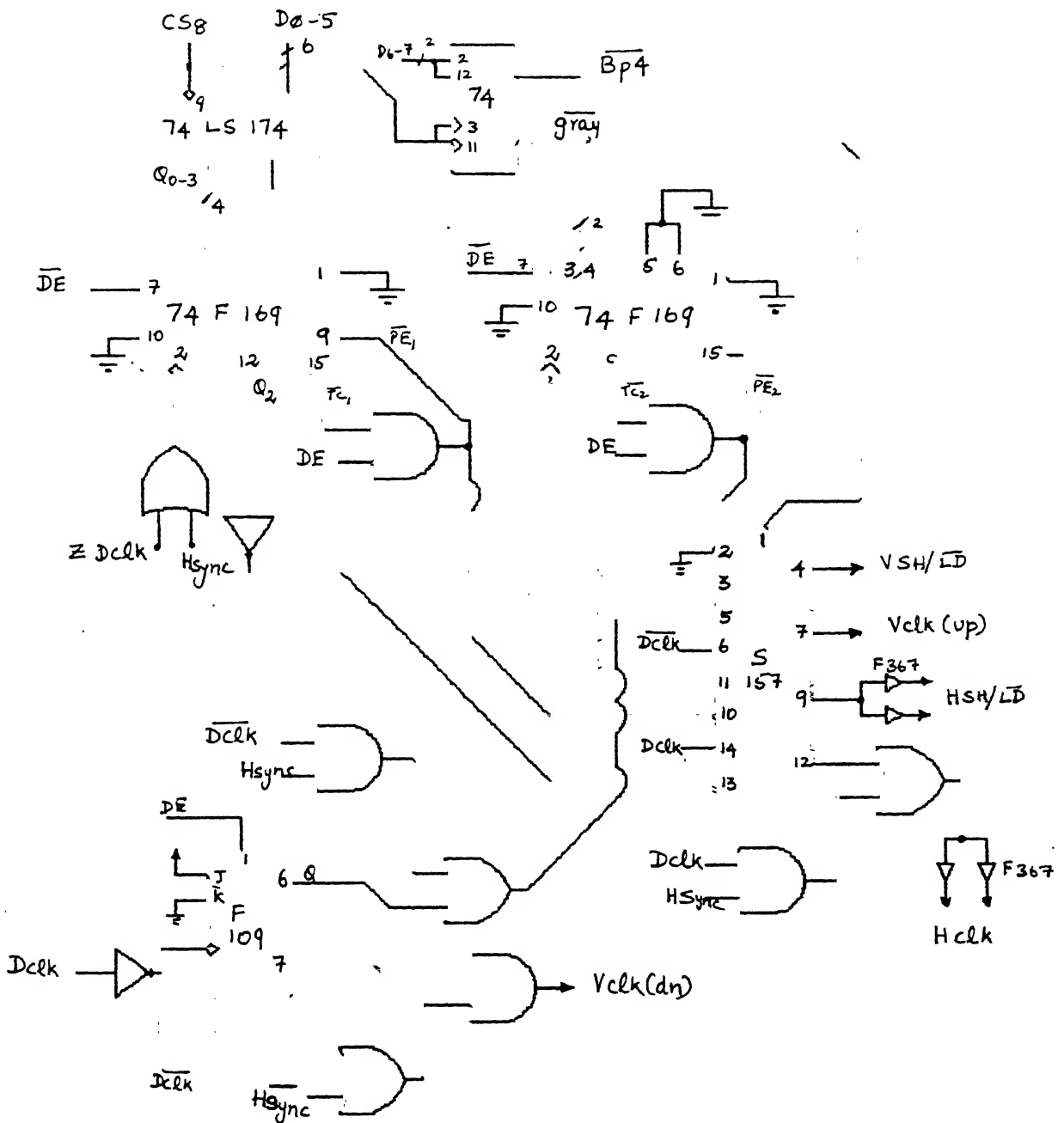


Fig 3.11. CONTROL CIRCUIT FOR PS.

made up of 2 bits which determine whether display mode is monochrome or gray-shade; if later then other bit decides whether Bpp is 4 or 8. The Bpp register 74LS174, is a hex D-type flipflop. The first four bits of which decide the division factor for the counter-1 and remaining 2 bits decide that of counter-2.

Consider HSH/\overline{LD} in Table 3.1, to obtain this signal it has to be divided by one of the factors 4, 8, or 32. Since F169 is a 4-bit counter, a maximum of divide-by-16 operation is possible, therefore to divide by 32 two counters are required. In the same 1-Bpp case where the division by 32 is required, a $Dclk \div 8$ signal is also required for VSH/\overline{LD} . Both of these can be obtained using two counters, and by programming counter-1 to divide-by-8 and feeding this divided clock to counter-2 which has a division factor of 4. It is clear from the Table 3.2 that for cases $Bpp > 1$, the division factor never exceeds 16, in fact it is a maximum of 8. So in these cases one counter is enough. Accordingly counter-2 is enabled only for monochrome configuration. The different values to which the counters are to be programmed depending on the required Bpp is given in Table 3.2.

As already discussed for zoom control circuit, for initial loading of counters 1 and 2, the clock input to counter-1 is ORed with Hsync and its \overline{PE} is controlled by DE. As the \overline{TC} will have decoding spikes, the counter-2 clock input is fed from Q_2 instead of \overline{TC} . The shift registers also have to be pre-loaded during the retrace period so the SR pipeline is full with correct pixel data when

Table 3.2 : Counter preset values

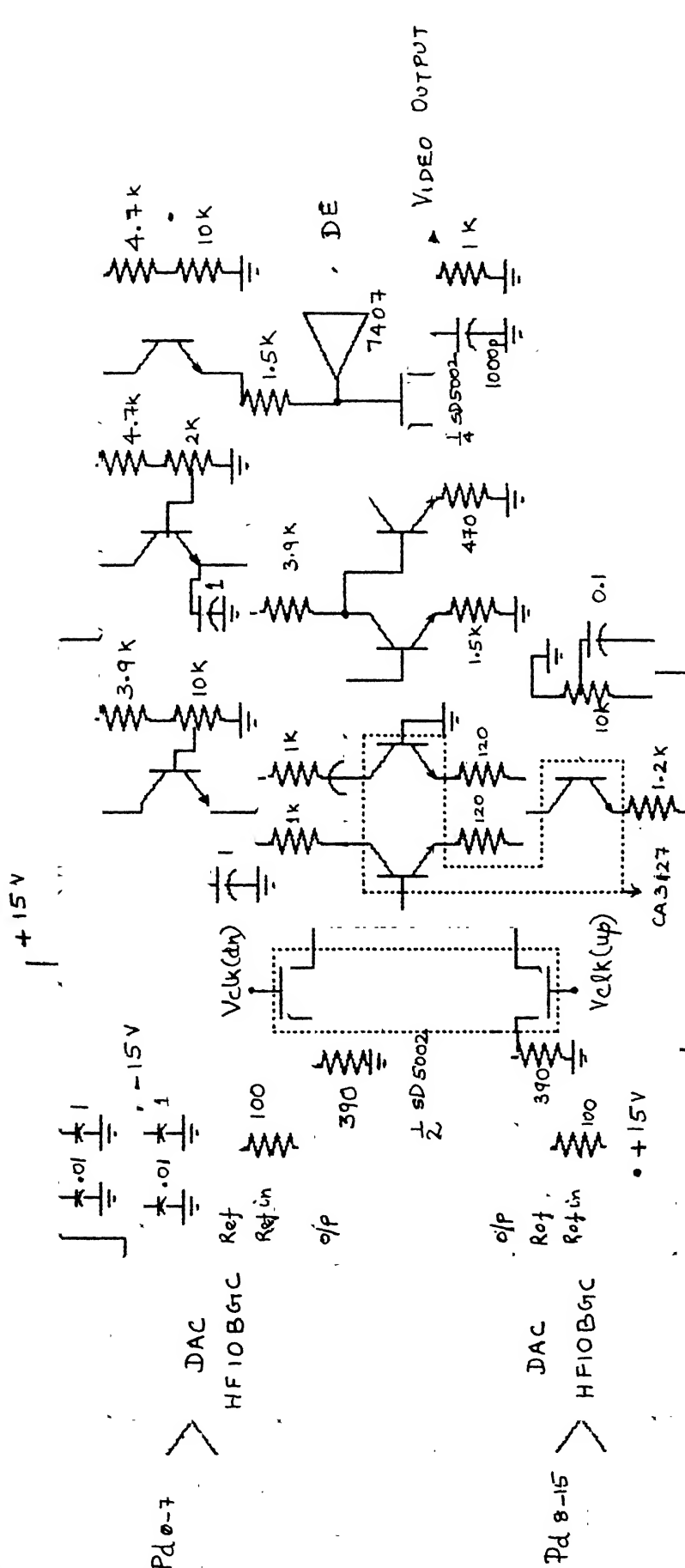
	Counter_1	Counter_2
Bpp=8	3	-
Bpp=4	7	-
Bpp=1	7	3

rising edge of the 1st dot clock arrives, hence the $\bar{P}\bar{E}$ of counters 1 and 2 are given to MUX instead of $\bar{T}\bar{C}$.

The $\text{Volk}(\text{dn})$ is obtained by dividing the ZDclk using a toggle flipflop which is cleared to a predetermined value by DE during start of every scanline. During zoomed conditions, Dclk gets divided by the Dclk divider, which during the retrace period maintains ZDclk low. But the shift registers and latch should get preloaded before the starting of scan line, which essentially requires some rising edge during retrace period. This is achieved by various ways for Hclk , $\text{Volk}(\text{up})$ and $\text{Volk}(\text{dn})$ as shown. As the Hclk and $\text{HSH}/\bar{\text{LD}}$ should operate at a maximum clock frequency of 50 MHz and they have to drive 8 shift registers, therefore they are buffered using 74F367. The timing waveforms pertaining to this control circuit have already been given in Fig 3.8.

3.9 Video Generator

Any Video Generator (VG) design will depend on the specification of the monitor which it has to drive, as it forms the direct interface between the PS and CRT display. In this implementation, VG consists of DACs, a differential amplifier, a Common Emitter (CE) amplifier and a emitter follower, and provides a linear voltage variation from 1 to 5 volts for its digital inputs correspondingly ranging from 00 to FF (for $\text{Bpp}=8$). The latches L_1 to L_4 serve to hold the digital data for the DAC. The DAC, DAC-HF10BGC, is a 10-bit current output device. The maximum output settling time is 25 nanoseconds. They can directly drive a resistor load for upto ± 1 volt. It has an output voltage compliance of ± 1.2 volts. It requires +15V and -15V power supply. The connection diagram of the two DACs are shown in Fig 3.12. The output current of these is passed through a 390 Ω resistor, which acts as a current to voltage converter, directly to produce a voltage swing of 0 to 1 Volt corresponding to a digital input from 00 to FF. The analog switches



-15V

ALL RESISTANCES ARE IN OHMS
ALL CAPACITANCES ARE IN FARADS
ALL DISCRETE TRANSISTORS ARE 2N2219.

Fig. 3.12. VIDEO GENERATOR

SD5002 are connected in such a way that they act like Analog MUX. SD5002 is a quad array of n-channel enhancement-mode DMOS FET, with a threshold voltage of 2V and T_{on} of 1 nanoseconds. It can handle a maximum analog signal voltage range of ± 7.5 V.

The transistors in the differential amplifier are from a single transistor array CA3127, having 6 transistors. It amplifies and gives a dc offset of 1 Volt to the signal so that the following CE stage can amplify without distortion. This is followed by an emitter follower which provides enough current to drive the CRT directly. There is a switch at the end which ensures that no video signal is passed through to CRT during retrace period.

The voltages V_1 , V_2 and V_3 are obtained using different potential divider arrangements. These voltages are to be varied to control gain and dc offset.

CHAPTER 4

VIDEO UPDATE MODULE

The task of the Video Update Module is to generate and provide the necessary address to FB, both for updating and for normal refresh of video display. An in-depth analysis of how these tasks are incorporated in FGA following a radically different approach, is given in this chapter. This chapter also discusses the bus interface unit which calls for stringent drive requirements and other critical operating parameters in a VME bus environment.

4.1 Relevant Features of VME Bus

Before the design of various functional units are taken up, it is worthwhile to take a glimpse of VME bus protocols. As FGA does not have an interrupter, only data transfer and bus arbitration cycles are discussed in the following sub-sections.

4.1.1 VME Read cycle

The read cycle timing diagram is shown in Fig 4.1a. The master can read 8, 16 or 32 bits in a single cycle by providing appropriate combinations of LWORD (Long Word Selection) and the Data Strobes DS1 and DS0 - e.g. to read a 16-bit word LWORD has to be negated and both data strobes asserted. The Address Modifiers (AM) specify the type of access - supervisory, normal, block, program, data etc. They also specify the number of address lines which will be used, depending on 16, 24, 32 bit addressing modes. For 24 bit standard non-supervisory

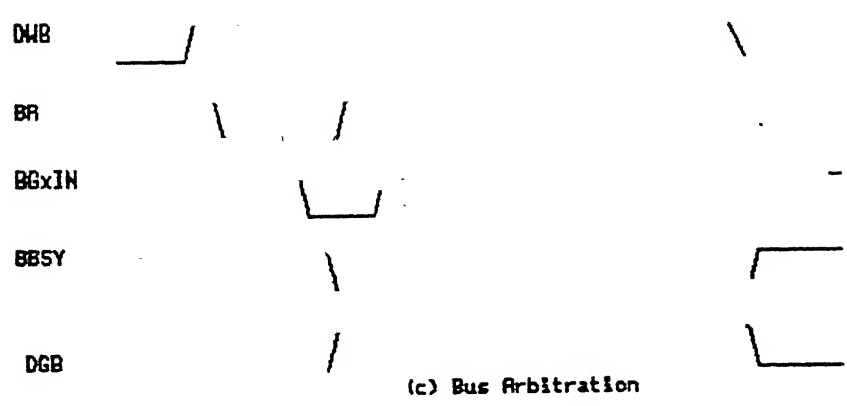
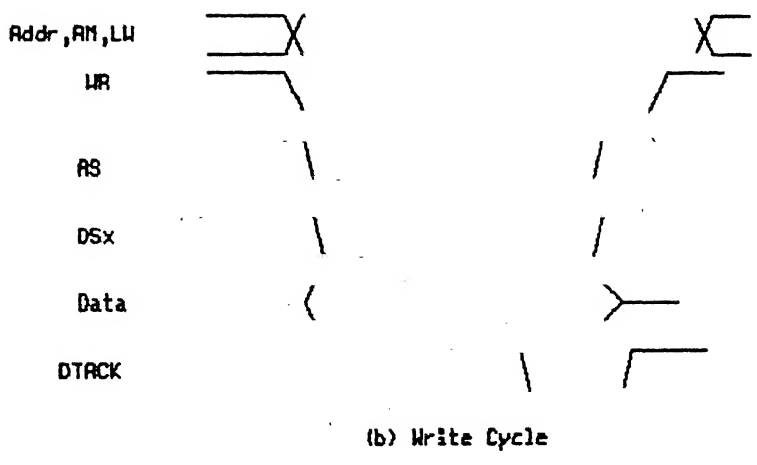
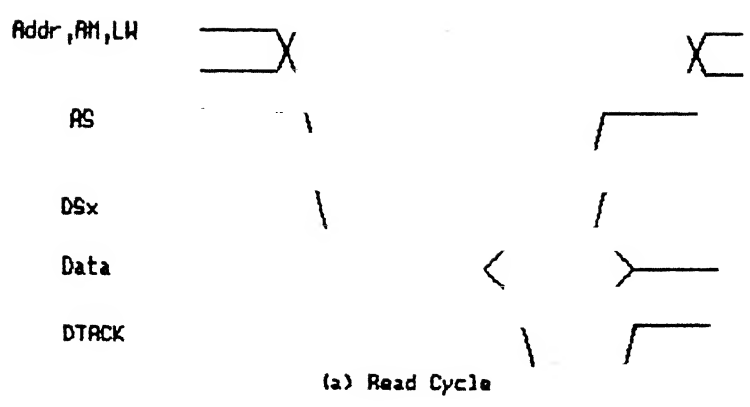


Fig 4.1 VME Bus Cycles

data access AM code is 111010.

To read data from a slave, the master in control of the bus initiates the read cycle by placing the address on the address bus and asserting AS to indicate that valid address has been placed. It then asserts appropriate data strobes to indicate which of the bytes is to be transferred. If a particular slave finds itself addressed, it places the data after a certain delay and asserts DTACK indicating valid data on data bus. On receiving the DTACK the master latches the data and negates the strobe signals, after which slave also can negate the DTACK, thus concluding the read cycle.

4.1.2 VME Write Cycle

The various signals and their sequence of assertion/negation to bring about a write is shown in Fig 4.1b. The Master places the address and AM code on the address bus and asserts AS. In the write cycle it then places the data on the data bus and asserts appropriate data strobes. The slave decodes the address and AM lines and if the write is addressed to it, receives the data and asserts DTACK. On receiving DTACK, master concludes that successful write has taken place and negates all the control signals it has driven. If due to some reasons, the slave is not able to accept the data sent by master it can assert BERR.

4.1.3 Bus Arbitration Cycle

When a potential master requires VME bus it sends a signal called the *Device Wants Bus* (DWB) to its bus requester. On sensing this, the bus requester asserts BR_x (x = 0, 1, 2, 3). The arbiter sitting on the first slot senses this request and issues a BG_xIN signal which propagates through the daisy chain and arrives to the requested requester. As soon as the bus requester gets this signal, it asserts BBSY, indicating that the new master has taken over the bus and

sets *Device Granted Bus* (DGB) flag and negates BRx. BGxIN is negated as soon as BBSY is asserted. The bus requester holds BBSY asserted till the time its local master asks it to release the bus by negating DWB signal. This protocol for acquiring the VME bus is illustrated in Fig 4.10.

4.2 VIDEO CONTROLLER (VC)

The video controller design, when given a superficial look, will appear simple as they just have to do simple synchronised address generation. The following discussions bring out the intricacies involved in the design of VC.

4.2.1 Address Switching in VC

The generation of refresh address for FB can be implemented using simple counters, but some issues have to be kept in mind before such an implementation. As VC just does a serial scan of FB, a n -bit counter for a FB of size 2^n longwords, is enough. But with such a design, problems crop up if hardware zooming has to be done. A simple and straight forward way to implement zoom is to separate the VC address into two groups - Horizontal and Vertical addresses. With every load signal of PS (HSH/ \bar{LD}) the horizontal address has to be incremented and after every Hsync vertical address has to be incremented. We then just have to slow down the HSH/ \bar{LD} and Hsync by appropriate factor to achieve zooming in X and Y directions. Hence the implemented VC has 2 counters - horizontal and vertical.

The statement *separate the VC address* is vague, but this has been purposely done to highlight the second design issue involved. The total number of address bits gets automatically fixed once the memory size is fixed. But to decide the separation - how many bits to be allocated on either side - requires careful analysis of the required addressing capability. For 100% flexibility, FGA should

be capable of providing the maximum resolution possible with a given memory size. For example if FB is 128k bytes then it should be able to handle 512 X 256 resolution with Bpp=8 and also 1024 X 1024 with Bpp=1. This may not be achievable if we have a fixed number of bits coming out from the horizontal and vertical counters. Moreover if the number of address bits are fixed, there will be a lot of wastage of memory, and a situation might arise wherein the memory is available for storage of a particular resolution image but the VC is not capable of addressing it. Hence a judicious choice has to be done regarding the separation of address bits into vertical and horizontal groups.

After careful analysis, for a FB size of 256k bytes (2^{16} LW), it has been found that the (8,8), (9,7) and (10,6) combination (V and H) of address bits can cater to the most of the available standard resolutions like CGA, VGA, HGC, EGA, Hi-Resolution, etc. Table 4.1 summarizes these combinations. If more flexibility in terms of resolution is required, increasing the memory size is the easiest solution.

4.2.2 VC Design

The combinations discussed above can be achieved using 3 different counters and three multiplexers (Fig 4.2a). 74F779 are 8 bit counters having bi-directional I/O pins. There is an additional 74F169 configured as a 2 bit counter.

Table 4.1 Address separation combinations				
Pixels/line	Bpp	Transfers/line	Lines/frame	Address (V+H)
640	8	160	200	8+8
640	4	80	350	8+8
640	4	80	480	9+7
1024	1	32	768	10+5
1280	1	40	1024	10+6
720	4	90	348	9+7

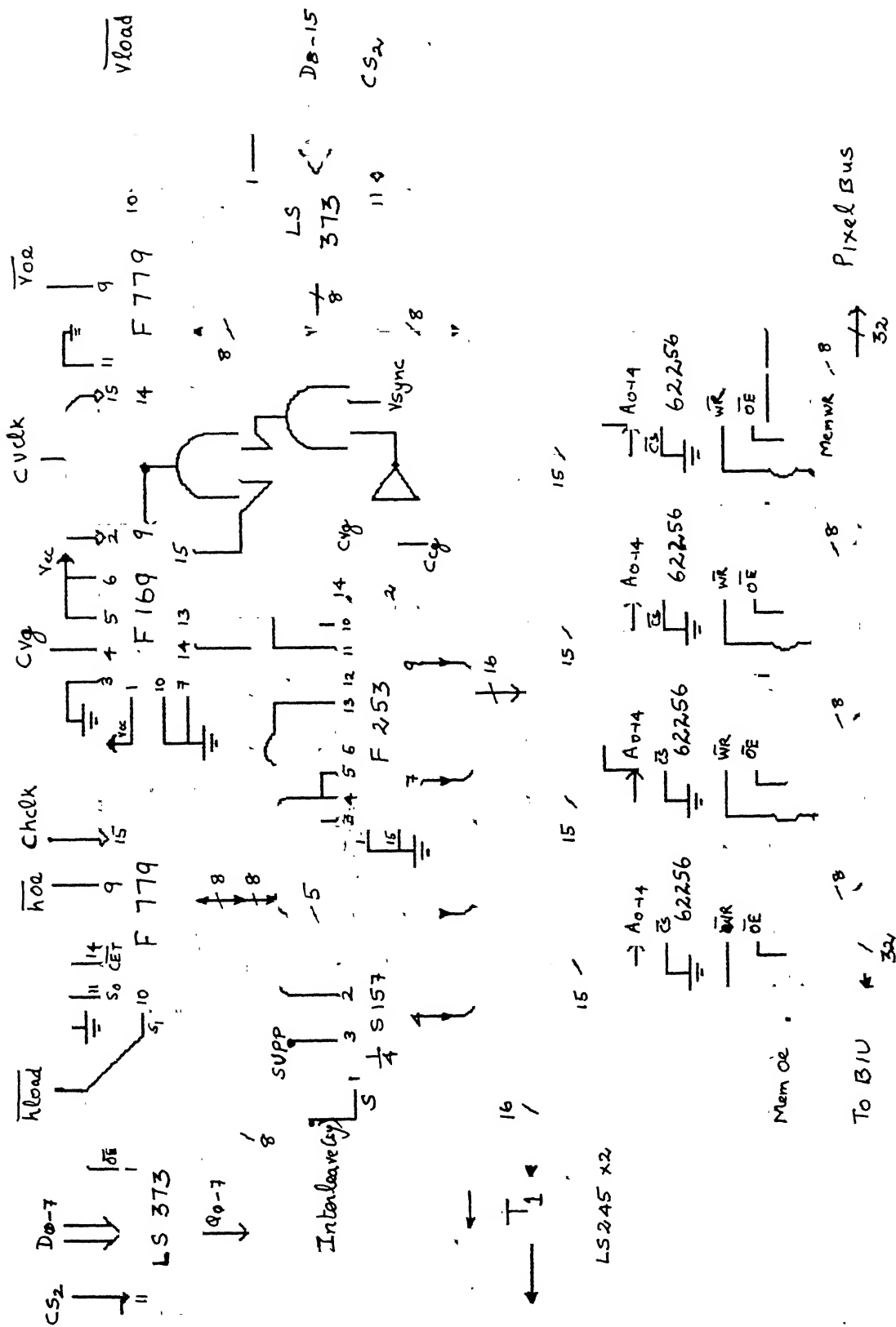


Fig 4.2 (a). VIDEO CONTROLLER.

When 10-bit vertical address is required then this 2 bit counter which is clocked by HSync is enabled. This condition is achieved if the control bits Ccg and Cvg are both zero. The other combinations of address bits achievable using these control bits is shown in Table 4.2.

The horizontal counters are loaded after every scan line by HSync be it zoomed or unzoomed conditions. They get incremented by HSH/ $\bar{L}\bar{D}$ during normal operations and with its delayed form when zooming takes place. The vertical counters get HSync as clock and these will not be incremented for some scan lines depending upon zoom factor, if zoom is enabled. All the counters are also initialized by Vsync. The control circuit to achieve the loading and counting of VC is shown in Fig 4.2b.

4.2.3 Simultaneous Scan Update Cycle

The memory contention problem was discussed in detail in Chapter 2. One of the aims of developing FGA was to try out a rational idea to avoid this contention. Almost all the design till date have tried to optimize the time available for refresh and update access in order to minimize the disruption involved in picture generation. The main reason for the memory contention is due to the fact that update and refresh access are two events which tend to occur simultaneously, and both these events have to be serviced immediately. FGA has been designed in such a way that memory contention problem is not allowed to

Table 4.2 VC Address Switching

Ccg	Cvg	H	V
0	0	16	10
0	1	7	9
1	0	8	8

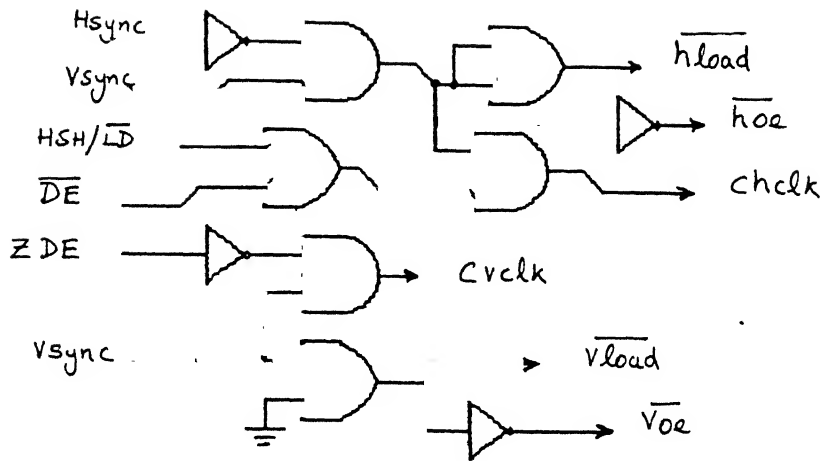


Fig 4.2 (b) CONTROL CIRCUIT FOR VC

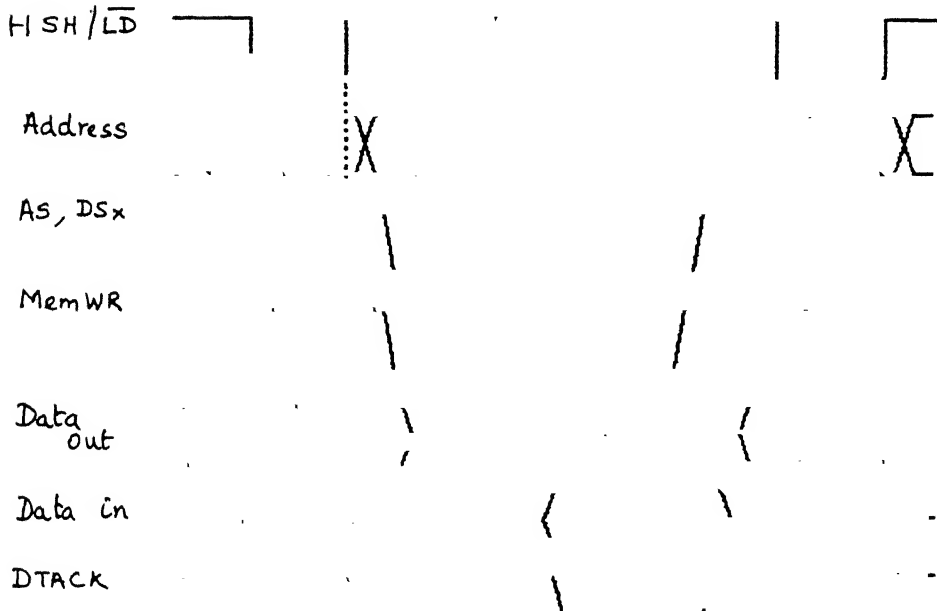


Fig 4.3. SIMULTANEOUS SCAN UPDATE CYCLE

occur. This has been done by attending to the above two events simultaneously, with a memory cycle named Simultaneous Scan Update (SSU) cycle. In this method the VC not only provides refresh address but also provides the update address. When FB has to be updated the transceiver T_1 (Fig 2.7) is enabled and the VC address is placed on the VME bus in A2 to A17. VC is capable of providing only 16 bit address, while VME bus requires standard addressing; the upper order address bits (A18 to A23) and AM codes are provided by the DMAC (to be discussed in next section) which is then the bus master and as such drives all necessary control lines. This address goes to the system memory and a read cycle is performed, resulting in the loading of data into FB as well as into PS. This means that the pixel bus becomes an extension of the VME data bus during the update cycle. Thus the data that gets written also gets displayed immediately. As the generation of address is synchronized with respect to HSync, VSync and HSH/ \bar{LD} , the read from system memory has also to be synchronous. Thus the asynchronous VME bus is, as it were, made to operate in a synchronous fashion and do the memory transfer, with the system memory access time decided by FGA. The access time of system memory with the present design of FGA has to be less than 300 ns, which is not an unreasonable demand with present memory technology providing access times less than 100 ns. The timing waveform of SSU is shown in Fig 4.3.

4.2.4 Update by Interleaved Transfer

Updating the full screen with SSU cycles, requires an entire frame time. Sometimes, with high-resolution gray-shade display, FGA demand on the system memory access time might be unreasonable. For FGA to work in such cases an optional interleaved transfer mode has been provided. In this by allowing the displayed image to get slightly disrupted, the memory-to-memory transfer takes place in two frame times. During the first one the A_1 bit from VC is suppressed

to 0 and all even locations are updated; during the second frame A_1 bit is held at logic '1' and all odd locations are changed. The suppression of A_1 is achieved by a multiplexer (Fig 4.2a).

4.3 Direct Memory Access Controller (DMAC)

The major functions of the DMA controller are - i) to acquire the bus strictly following the VME protocols, ii) to supply the page address from where the data has to be transferred and iii) to provide the necessary data and address strobes for the VME bus as well as MemWR signal for FB. These functions are implemented by the circuits discussed in the following sub-sections.

4.3.1 Bus Requester

The implementation of the bus requester is shown in Fig 4.4. The DWB signal comes from a flip-flop which is set, by an appropriate chip select signal designed by the Bus Interface Unit from the address sent by the host processor. The BGOIN is given as a clock to the flip-flop generating BRx, provided AS is high. This is done to ensure that new master does not start driving the bus before the previous master has released AS. The DGB signal, after it is received, is synchronized with Vsync so that DMA can start exactly when the frame starts. A counter clocked by Vsync provides a bit Q_0 which suppresses A_1 of VC when interleaving has to be done. It is true that with interleaved transfer mode SSU cycle can no longer be simultaneous in the strictest sense, but this is the trade-off for low-end systems.

4.3.2 Page Address Register

The register which provides the page address has been implemented with two 74LS373's (Fig 4.5). This latch has to be programmed by the host processor with

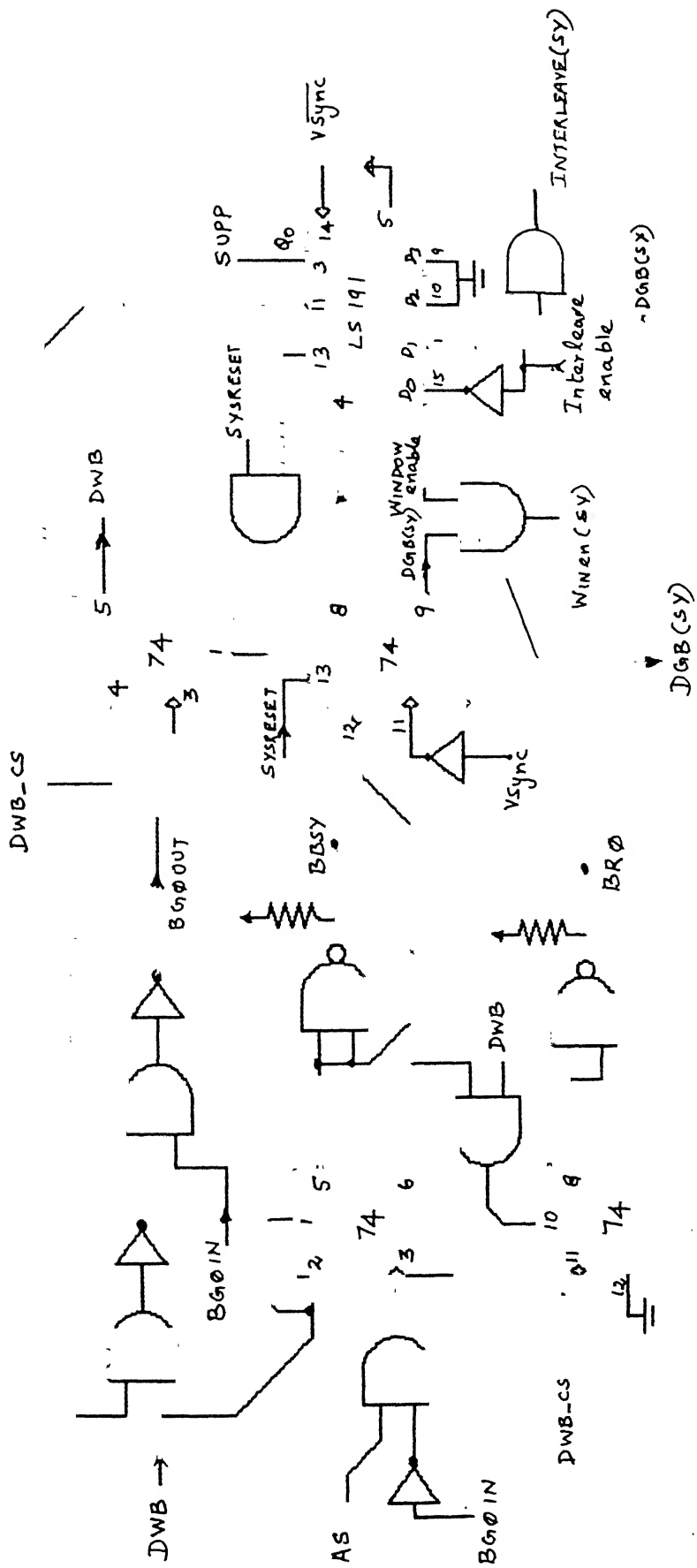


Fig 4.4. Bus REQUESTER

28-15

D0-7
DnB(sy)
CS1

D0-7

LS 373

LS 373

ADDRESS REGISTER

6
A23-18 A1

6
A0-5 WR LW

Fig 4.5 PAGE

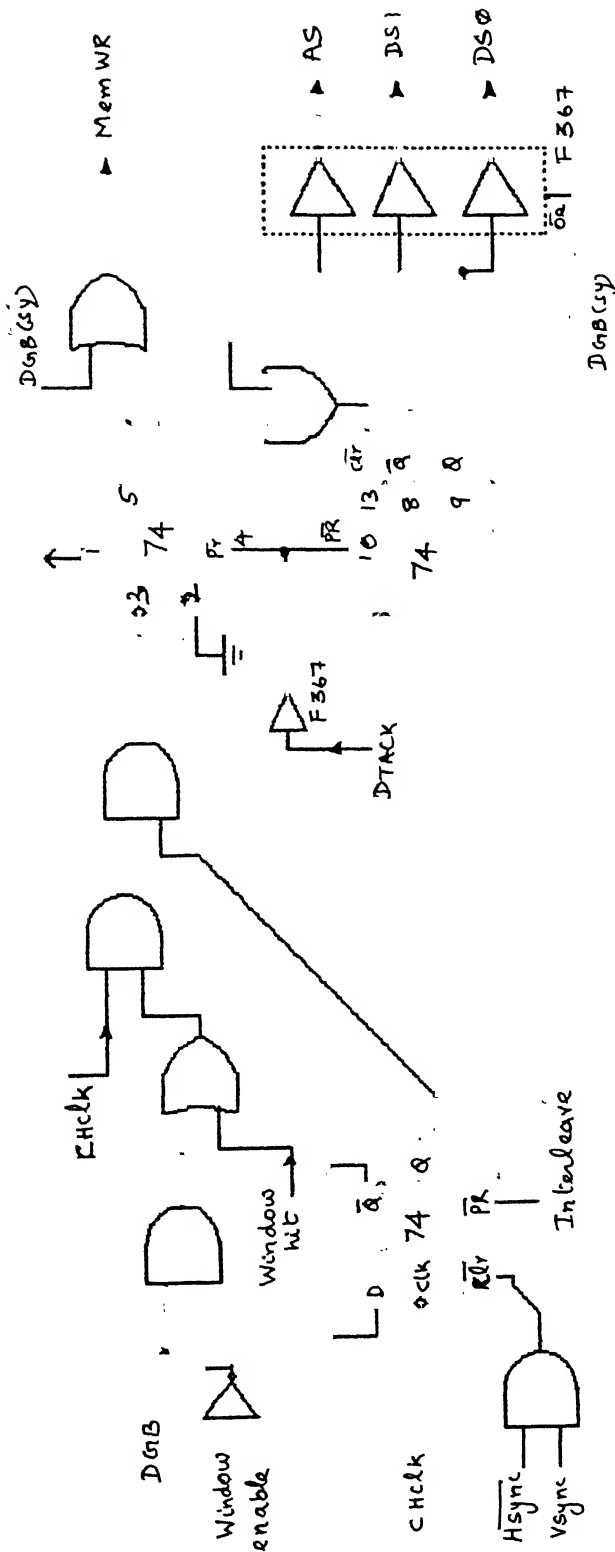


Fig 4.6. MemWR AND STROBE GENERATION CIRCUIT

the page number of system memory from where the image has to be transferred. The latch output is enabled when the device has been granted bus. In the present design the page address register provides A18 to A23 in addition to AM codes. Since the DMAC always has to read a longword for a DMA transfer, the signals A₁, LW and WR can also be provided by a register. So the Page Address Register is programmed to drive A₁=0, LW=0 and WR=1 in addition to other bits.

4.3.3 Strobe Generator

As DMA transfer is fixed at 32 bits, the signals AS, DS1 and DS0 can all be asserted simultaneously in a VME read cycle. So these signals are internally derived from a single source (Fig 4.6). MemWR signal can also be derived from these signals. In the implemented circuit the first flip-flop ensures that during interleaved transfer, AS is strobed in alternate cycles.

4.4 Window Manager

Referring to chapter 2, the 82786 Graphics Coprocessor implements hardware windows and can display a limited number of windows. The 82786 accesses the display list structure, which holds the context information of different windows, during every frame. It does comparison between the memory refresh address and the value given in the display list, to find out whether the window boundary has been reached; if so, it starts displaying a different bit map indicated in the display list. This process is repeated frame after frame. As the user response is rather slow, the windows will be altered or created with time intervals which are many orders greater than frame time. Hence it is a wastage of resource to repeat the same comparison process over and over again for many seconds. In the proposed method this is overcome by creating windows actually in the frame buffer itself and this window boundary calculation has to

be made only once that is when the window is copied. In this scheme windows will be in separate bitmaps in main system memory and they will be clubbed together in the frame buffer, thus maintaining one to one correspondence between the screen the frame buffer always. This clubbing is achieved by doing a selective write in the specified window area of FB.

Window Manager has 2 major functional units – Window comparator (Womp) and Window Address Generator (WAG). These two are taken up in the following sub-sections.

4.4.1 Window Comparator

Womp (Fig 4.7a) has latches which have to be programmed with the required window parameters like size and position. Womp generates *Window Hit* signal, when the raster scans the specified window area. To generate this hit signal the Womp utilises two sets of counters – horizontal and vertical. The horizontal counter(74F779) gets loaded with two values provided by two latches. The first latch (WL1) contains the X co-ordinate of the top left-hand corner of the window in terms of number of longwords. The other latch WL2, specifies the width of window in terms of number of longwords. This means that the horizontal extremities of windows have to be on longword boundaries. WL1 is loaded in the horizontal counter by the Hsync and WL2 is loaded after the first boundary is detected. This process continues for every scan line of the frame in which window DMA has to take place.

The 10-bit vertical counter is made up of 74F169 and 74F779. It also has two latches WL3 and WL4, specifying the upper Y co-ordinate and the height of the window respectively. WL3 gets loaded by Vsync while WL4 gets loaded after the first vertical boundary of window has been encountered. The $\bar{T}\bar{C}$ signals coming out from the horizontal and vertical counters pass through a logic which

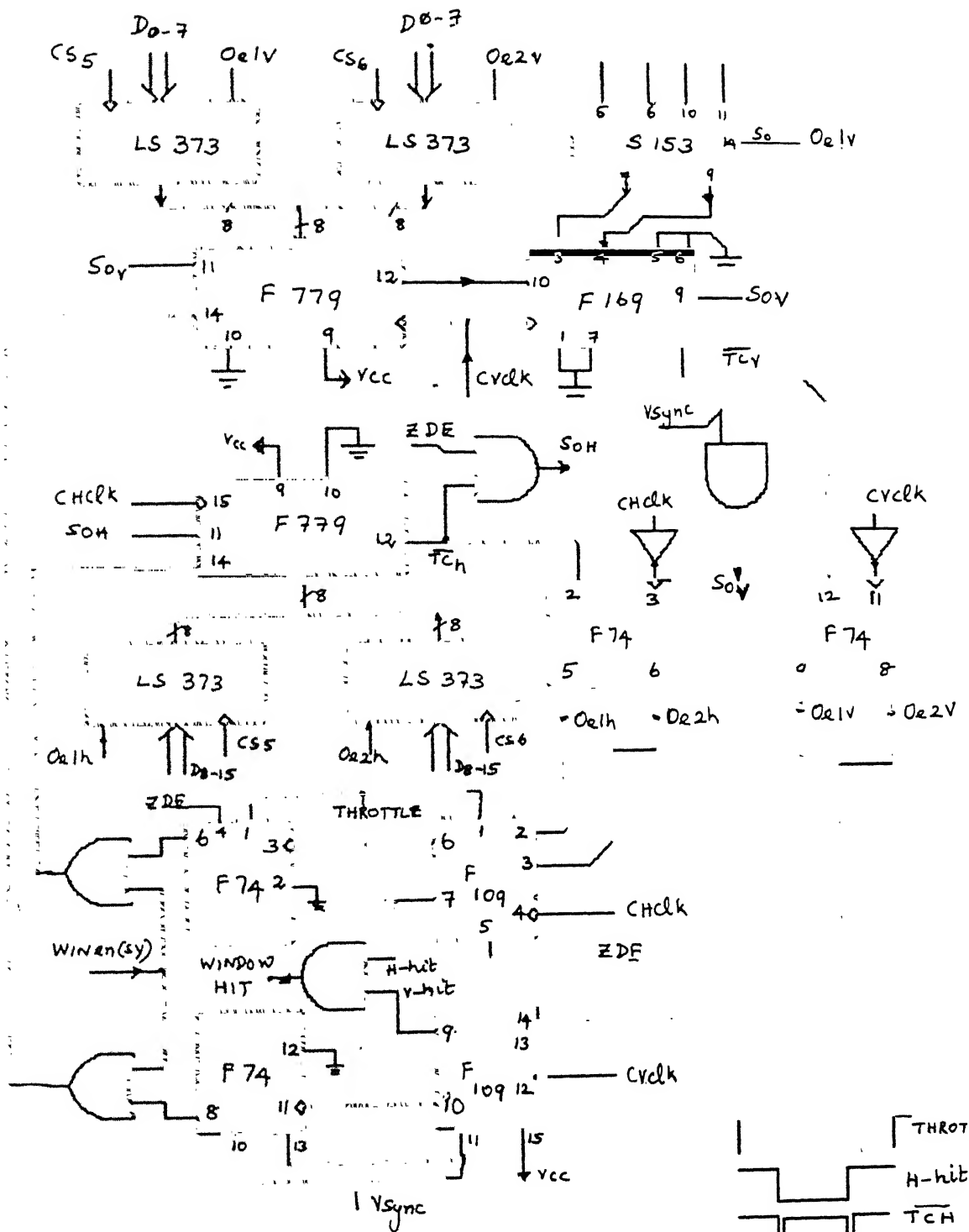
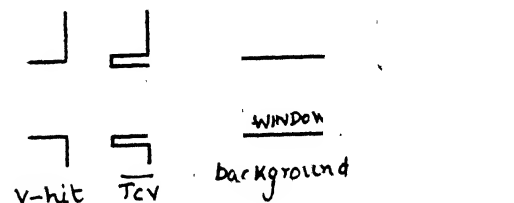


Fig 4.7(a) WINDOW COMPARATOR

Fig 4.7(b) WINDOW-HIT WAVEFORMS



gives out the window hit signal (Fig 4.7b).

4.4.2 Window Address Generator

Whenever window hit signal is true WAG (Fig 4.8) increments and places address on the VME bus and an SSU cycle is performed, thereby changing the specified window portion alone. The control circuit of WAG is shown in Fig 4.8b. It should be carefully noted that the window address is not used for addressing FB, as the transceiver T_1 is disabled during window write. The WAG is implemented using a continuously running counter, unlike in VC where the address was separated into vertical and horizontal bits. Because of this the window contents can be stored continuously in the main memory, irrespective of the screen resolution. This will not be the case if VC is used for DMA where the processor has to store the image in X-Y pattern as seen in the screen, and some of the locations may be skipped while DMA if these do not fall in the visible portion of the raster. Because the window address is not split into two groups, actual window contents cannot be written when the zoom is enabled.

4.4.3 Additional Advantages of Windowing

The clear advantage of hardware windows is their high speed. By doing minor modifications both in hardware and software, additional benefits can be reaped. With the implemented SSU cycle it requires full frame time of 20 ms (for a frame refresh rate of 50 Hz), to update the whole displayed portion of FB. During this entire period DMAC holds the VME bus and releases the bus only after a frame time. This period is twice i.e. 40 ms in case of interleaved transfers. In some systems holding the bus for such a long time may become critical, and will be helpful if the transfer time is optimized. So a special mode of transfer, termed as the *Throttle mode*, has been implemented to do DMA of a

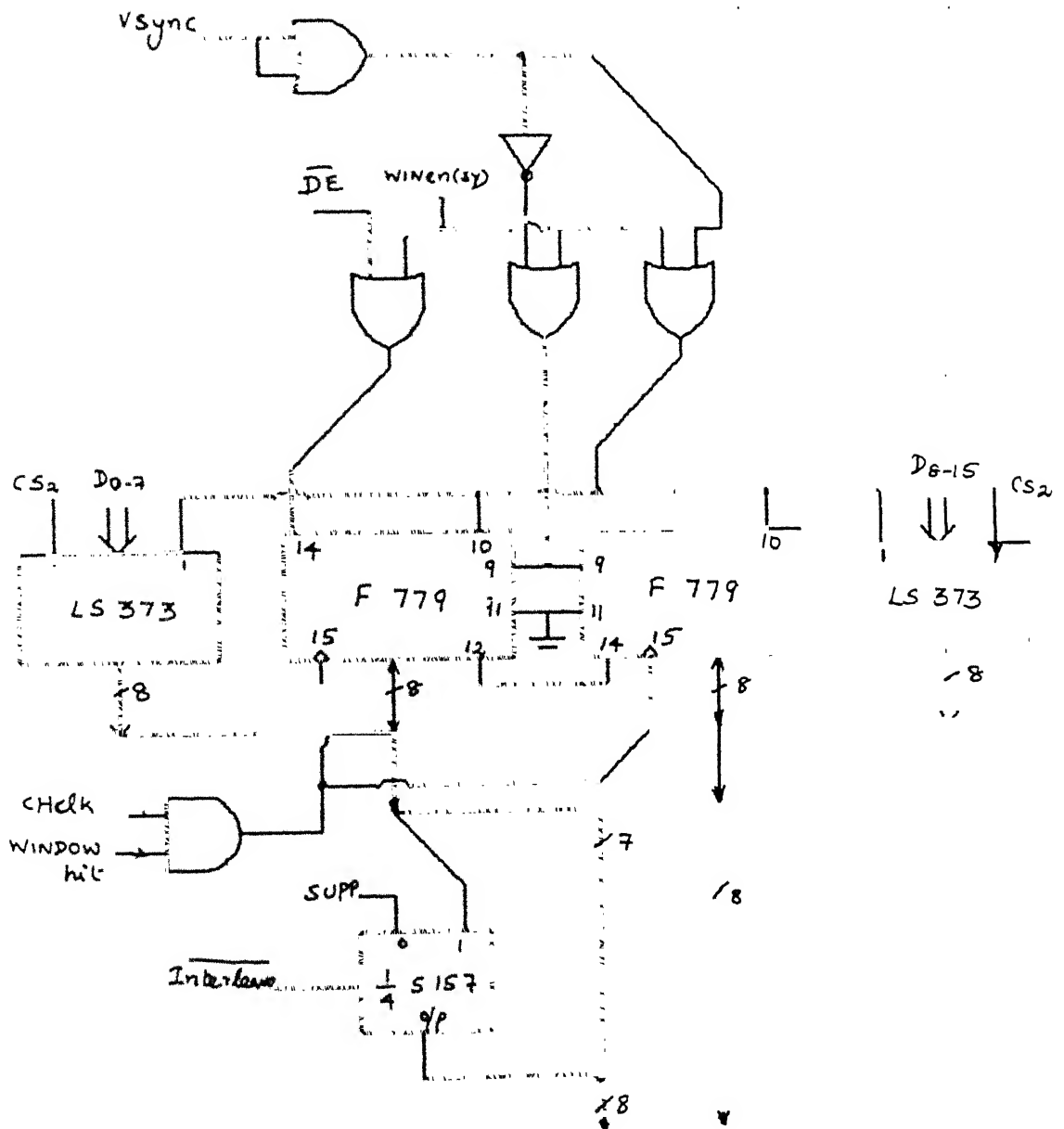


Fig 4.8 WINDOW ADDRESS GENERATOR

part of frame and then release the bus immediately. This mode can be used effectively if only a portion of image has changed and this has to be transferred, e.g. if in a 256 line display the top 4 lines have to be updated then using throttle mode it will take $4 \div 256 = 1/64$ of the frame time to do the transfer, thus increasing the efficiency to a great extent. But in the same example if the last 4 lines have to be transferred then the bus will be held for the entire frame time. The throttle mode has been implemented by suppressing the horizontal counter output of Womp, so that this signal becomes low throughout the scan line (Fig 4.7b). The window write can also be used to write small portions in the screen like cursor, for this the entire FB need not be rewritten.

4.5 Bus Interface Unit

Bus interface to VME requires bus drivers/transceivers which provide enough sourcing and sinking current capability to drive a signal onto the VME bus. The address decoder is also a part of BIU and provides chip select signals to various registers which are to be initialized. The various functional sub-units along with bus drivers are shown in Fig 4.9. These are covered in the following sub-sections in detail.

4.5.1 Address Decoder

The address decoding logic which generates various chip selects is as shown in Fig 4.10a. When DMA is not taking place, T_1 is disabled and the address comes to address decoder from the VME bus. The Interrupt Acknowledge signal IACK has to be decoded as VME rule states that slave should not respond to DTB cycles when IACK is low. If the Address Modifier lines AM0 and AM1 are not both low, then the AM values on the bus represent reserved AM codes and the decoder should not respond. This is ensured by appropriately decoding AM0 and AM1.

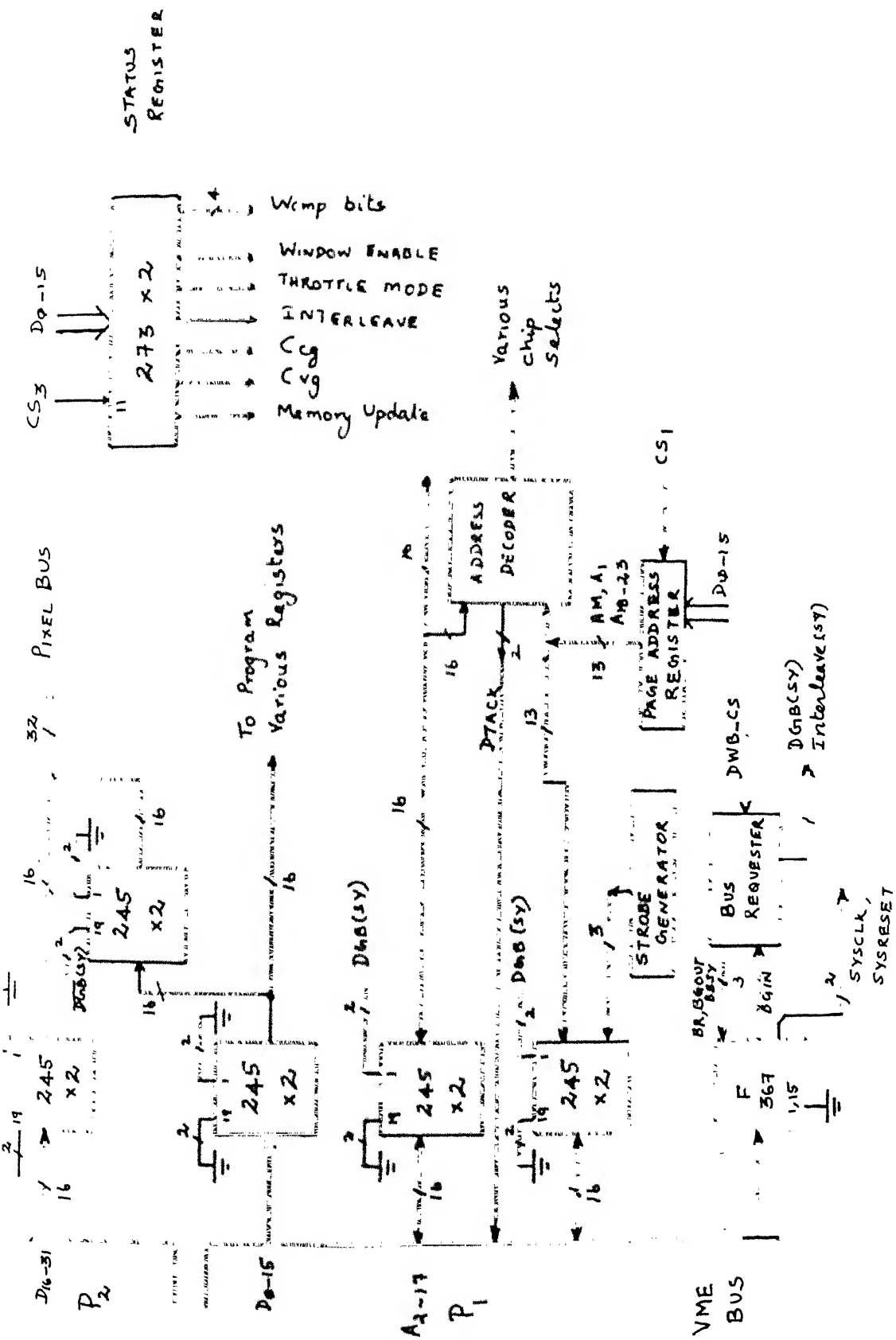


Fig 4.9. BUS INTERFACE UNIT.

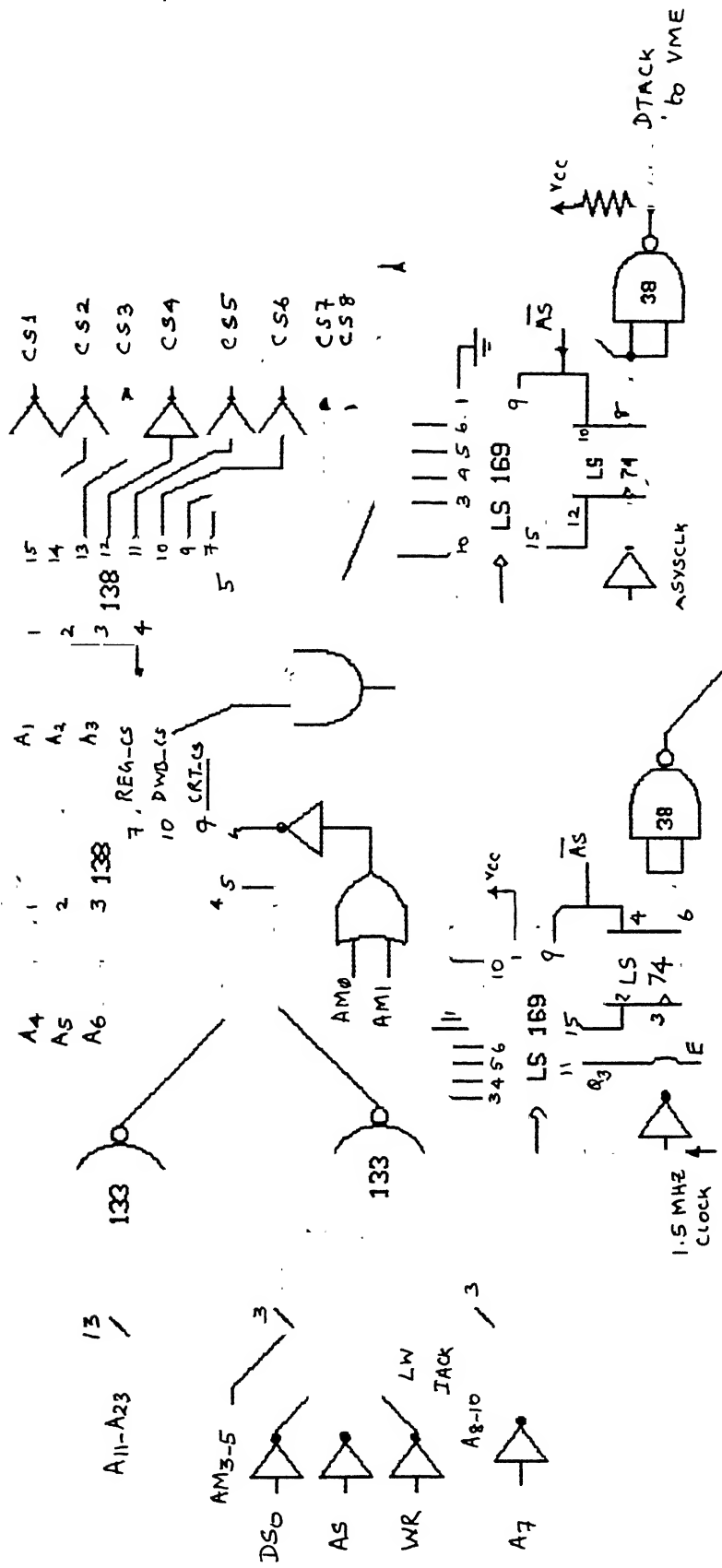


Fig 4.10(a) ADDRESS DECODER

Fig 4.10(b) CS WAVEFORM.

Other AM codes are decoded in such a way that FGA as a slave will respond to standard addressing mode (24 bit address) either for supervisory or normal access. The slave is configured as D_{16} slave for registers and $D_{08}(0)$ for the 6845 CRTC. BIU does not have the capability, in the present design, to assert BERR in case it is asked to respond as any other type of slave. For example, if a D_{32} slave mode transfer is attempted BIU does not respond with the DTACK and it is the Bus timer's job to assert BERR. FGA as a slave also cannot accept unaligned transfers. The various registers which are to be initialized are mapped in the address space 0xffff70 to 0xffff7f; if one of this is selected Reg_CS is asserted. The complete address map of FGA in present design is listed below -

Chip select	Address (in hex)	Used for
DWB_CS	0xffff50 to 0xffff5f	Device Wants Bus flag
CRT_CS	0xffff60 to 0xffff6f	CRTC 6845
CS1	0xffff70	Page address register
CS2	0xffff72	Window Address Generator (WAG)
CS3	0xffff74	Status register
CS4	0xffff76	Video Controller (VC)
CS5	0xffff78	Window Comparator 1 (Wcmp1)
CS6	0xffff7a	Window Comparator 2 (Wcmp2)
CS7	0xffff7c	Zoom factor register
CS8	0xffff7e	Bpp latch

4.5.2 DTACK Generation Circuit

There is a finite amount of delay required between the assertion of data strobe by master and response in the form of DTACK from the slave. This requirement is based on two reasons - i) to satisfy the VME rule which specifies

this delay to be a minimum of 30 ns and ii) to ensure the necessary setup time of various registers for them to get programmed correctly. The delay is achieved using two counters (Fig 4.10b), as this time is significantly different for 6845 programming from initializing the latches like 74LS373. Counter-2 is used for Register DTACK generation and is therefore controlled by DWB_CS or Reg_CS. The system clock from the VME bus is used for counter-1 and Counter-2 is clocked using an onboard 1.5 MHz clock. The $\bar{T}\bar{C}$ (Terminal Count) of counters, in both cases is fed to a D flip-flop which latches this low value and DTACK is kept asserted till AS is negated. As $\bar{T}\bar{C}$ is prone to decoding spikes it is not used directly to do asynchronous reset of flip-flop in both the cases. The DTACK is driven onto the bus by an open-collector gate (7438). An important point to note is that DTACK generated using Reg_CS is fed back to enable the second 74LS138, which decodes the lower address bits A_1 to A_3 . This is done so that, when a register is selected the output of second decoder is pulsed in a manner different from the normal decoder output. This pulsing is required for 74LS373 as they are level triggered latches and not required for registers which latch the data on a edge, like 74LS174. The generated chip select waveform is shown in Fig 4.10c.

As the programming of 6845 is synchronous and the write cycle time specified is 1 μ sec; the clock given to counter-2 is of low frequency. The various control signals generated for 6845 programming is shown in Fig 3.9c.

4.5.3 Status Register

The status register provides various control bits required for the operation of both update and refresh logic. The bit assignments of the status register follows -

Bit No	Usage	Active Logic
0-1	Window comparator 1 address (bit 16 & 17)	NA
2-3	Window comparator 2 address (bit 16 & 17)	NA
4-8	not used	
9	Enable window transfer	positive
10	Enable throttle mode transfer	negative
11	Enable interleaved transfer	positive
12-13	Cvg, Ccg control bits	NA
14	Enable memory update	positive
15	not used	

For hardware simplicity, some of the higher order bits required by the Window Comparators (Wcmp1 and Wcmp2), are also provided by this register.

CHAPTER 5

SUPPORTING SOFTWARE

This chapter discusses the basic software required to test FGA. The implemented device driver and user routine are analysed here and the listing in C is given in Appendix 1.

5.1 Device Drivers

For each peripheral device in a UNIX system, there must be a device driver to provide the software interface between the device and the system. A device driver is a set of routines that communicates with the hardware device and provides a uniform interface to the kernel. It manages the flow of data and control between the user program and peripheral device.

The UNIX Operating System (OS) supports two device models – *Character devices* like line-printer and *Block devices* like disk drive. As the OS interface exists inbetween any user program and the device driver, the two models look alike to most user programs. When a user program requests a data transfer of some bytes between its memory and a specific device, the OS transfers control to the appropriate device driver. Each device driver has associated with it two fixed 8 bit numbers, referred to as its major number and minor number, which together serve as a pointer to the particular special file (Character or Block device). Two switch tables – Block device switch table and Character device switch table -- contain the entry point addresses of various devices. These are accessed by the OS when a device driver call is made by a user program.

5.2 Device Driver Routines

Similar to the opening and closing of a file, a device also can be opened and closed, as UNIX treats devices as special files. The two essential routines of a device driver are *xxopen* and *xxclose* routines, where *xx* is a mnemonic that refers to the device name.

The *open* routine is called each time a device has to be accessed for the first time. It prepares the device for I/O transfer and performs any error or protection checking. The *close* routine is called after the end of the last transaction with the device. It is responsible for any cleanup operation that may be required, such as disabling interrupts, clearing device registers and so on.

Calls associated with transfer are *xxread*, *xxwrite* and *xxioctl*. The first two are concerned with the data transfer to and from user address space. An *ioctl* call is generally made when hardware dependent functions, such as setting the data transfer rate or programming status registers, have to be performed. As *ioctl* can also transfer data to and from user address space, it can be substituted in the place of *read* and *write* calls.

5.3 Device Driver Implementation

To test FGA the device driver should be able to program all the registers, it should be capable of acquiring memory of size at least equal to frame buffer size and give start address of this location to FGA. The following sub-sections discuss the features of the device driver developed for FGA.

5.3.1 *fropen* and *fclose* Routines

The functions performed by the *fropen* routine are given by the following sequence of steps :

- a) Find the starting address at which the replica has to be stored.
- b) Find the physical address for the above starting address.
- c) Extract from this physical address the required page address.
- d) Initialize the Page Address Register.
- e) Clear all other registers.

In addition to this, programming of CRTC 6845 can also be included so that the CRTC is initialized automatically whenever the device is opened. In the present design, programming of 6845 is similar to initializing any other register.

The *fclose* in the present implementation does not perform any other useful operation except closing the special file. But if needed some operations like disabling 6845 can be incorporated.

5.3.2 Replica of FB in System Memory

The required storage in system memory for placing the copy of FB, to enable FGA to transfer it and display the same, is acquired by declaring a buffer *frbuf*. Because of a specific hardware requirement the size of *frbuf* has been defined to be 256K bytes, although FB size is 128K bytes. As pointed out in Section 4.2.3, the decision to have simultaneous scan update (SSU) cycle necessitates that the upper address bits are provided by the page address register during a SSU cycle which is static i.e. not incrementing. On the other hand, as discussed in Section 4.2.2 the lower address is provided by a counter in the Video Controller block. Therefore the replica of FB in system memory cannot cross a 128K byte boundary. Hence the device driver has to acquire a perfect 128K byte page, which is impossible as there is no provision in C to state the location of the memory to be allocated. This problem is solved by asking for a 256K byte page and using a portion of it which represents a 128K byte. So the spill over on either side of this page, which amounts to 128K byte is unused.

The above problem may be illustrated more clearly by considering an example. Let the address returned by the kernel, after allocating memory space for *frbuf*, be 0xB8654. Clearly this address does not point to the start of a 128K byte page. Because of the facts that for a longword transfer A_1 and A_0 are zero and that VC provides 16 bit addresses, the implemented hardware can do DMA only from a address having the last 18 bits to be equal to zero. Thus the memory space that can be used is only from 0xC0000, with a result, a portion of allocated memory from 0xB8654 to 0xBFFFF and some more near the end will be left unused.

5.3.3 *frioc1* Routine

In the device driver of FGA there is no *frread* and *frwrite* routines. Programming of all registers is done by *frioc1* which passes arguments like *cmd* (command), to identify which of the registers has to be programmed. The data to be written in the register is passed through the U-area.

For the purpose of testing FGA, it is decided to fill *frbuf* with an image which is stored in a file. This has been achieved by transferring chunks of 1024 bytes from user routine to the system area. The user routine opens the particular file which has to be dumped as replica, reads 1024 bytes and stores it in an area termed as *loobuf*. Then it makes an *ioc1* call to the device driver which copies 1024 bytes of *loobuf* to *frbuf*.

CHAPTER 6

RESULTS AND CONCLUSION

The main aim of the work has been to develop a board having a standard interface, which can handle gray-shade monochrome display. The other equally important issues were the reconfigurability of Bpp and resolution. The designed FGA has been demonstrated to be working successfully and has achieved the goals aimed at. The capability of FGA to create windows within a single frame time has also been demonstrated.

6.1 Specific Test Results

The assembled FGA with its VME bus interface has been tested by using, as host, an HCL Horizon-III workstation, which is a 68020 based machine having a math coprocessor and running 4.2 BSD UNIX. The software listed in Appendix 1 runs in this system. The written user routine dumps an image file into a particular portion of system memory by calling the device driver. This is then taken by FGA and displayed on the screen. At present no software has been developed for creating graphics directly on the screen through FGA.

The FGA has been interfaced to a Samsung monitor which has a video bandwidth of 18 MHz and a horizontal scan rate of about 18 KHz. With this monitor, it has been possible to display a digitized photograph on a 640 X 256 frame. As the maximum number of lines that can be handled by the available monitor is limited by its low bandwidth, the FGA could not be tested at higher-resolutions.

6.2 Implementation problems

Some problems have been faced due to the limitations of the workstation to which FGA is interfaced. The Horizon III, though based on a 32-bit microprocessor, does not have the capability to drive 32 bits data in the VME bus. So the SSU cycle which requires a 32 bit data path is not possible. To overcome this the FB has been curtailed to 16 bit data width, and these 16 bits are replicated when loading the shift registers to form 32 bits.

Though the memory access time of the host system memory is 250ns, an idle time, equal to RAS refresh time (100 ns), is required in between two consecutive memory accesses. Because of this, the memory cannot be accessed at its peak rate, and even the Interleaved Update is too fast for the system memory. The update has therefore to be done by using a lower dotclock rate (= 1/4 dotclock rate for display).

The FGA has been tested by dumping some image files into FB. The photographs of the resulting display is shown in Fig 6.1. The display has a resolution of 640 pixels in the horizontal and 256 lines in the vertical direction. The figure also shows the other capabilities of FGA like windowing, zooming etc.

6.3 Suggested Improvements

There are some inadequacies with the present design which can be corrected. Though the FGA can perform scrolling, the present scheme used for scrolling causes a wrap over of the display. A better scheme to perform scrolling can be implemented.

In some multiprocessing environments the bus traffic created by FGA can become considerable. If this becomes a handicap, then amount of DMA can be reduced by incorporating some on board computing power. The block diagram of

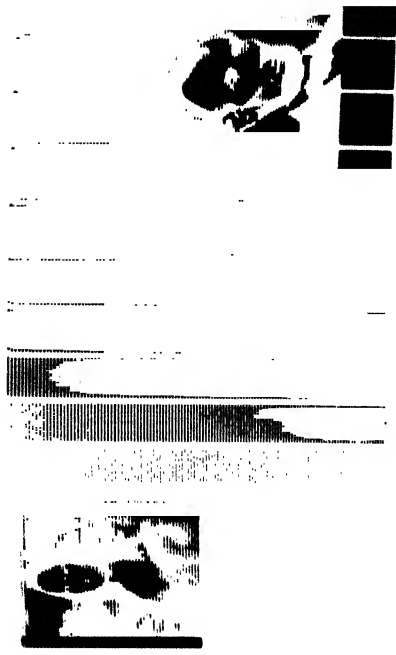
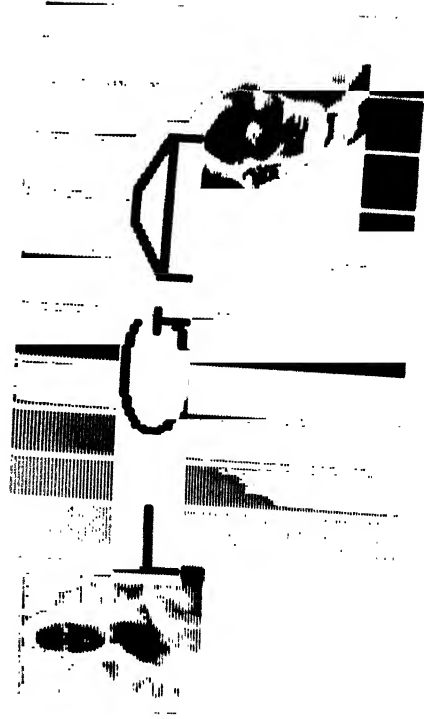
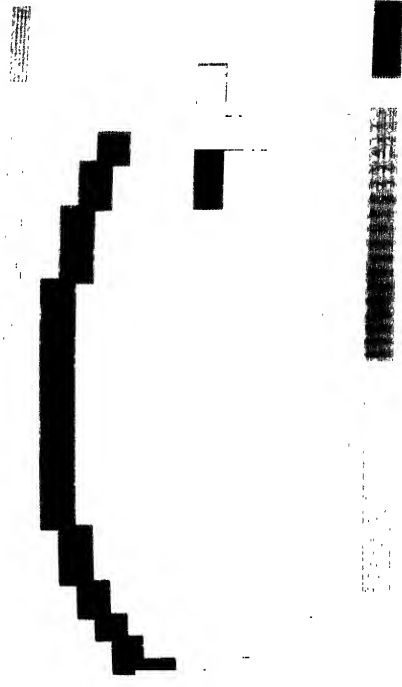


Fig 9.1(a) 640x256 display with two windows



9.1(b) WINDOW PORTION Zoomed 4 times



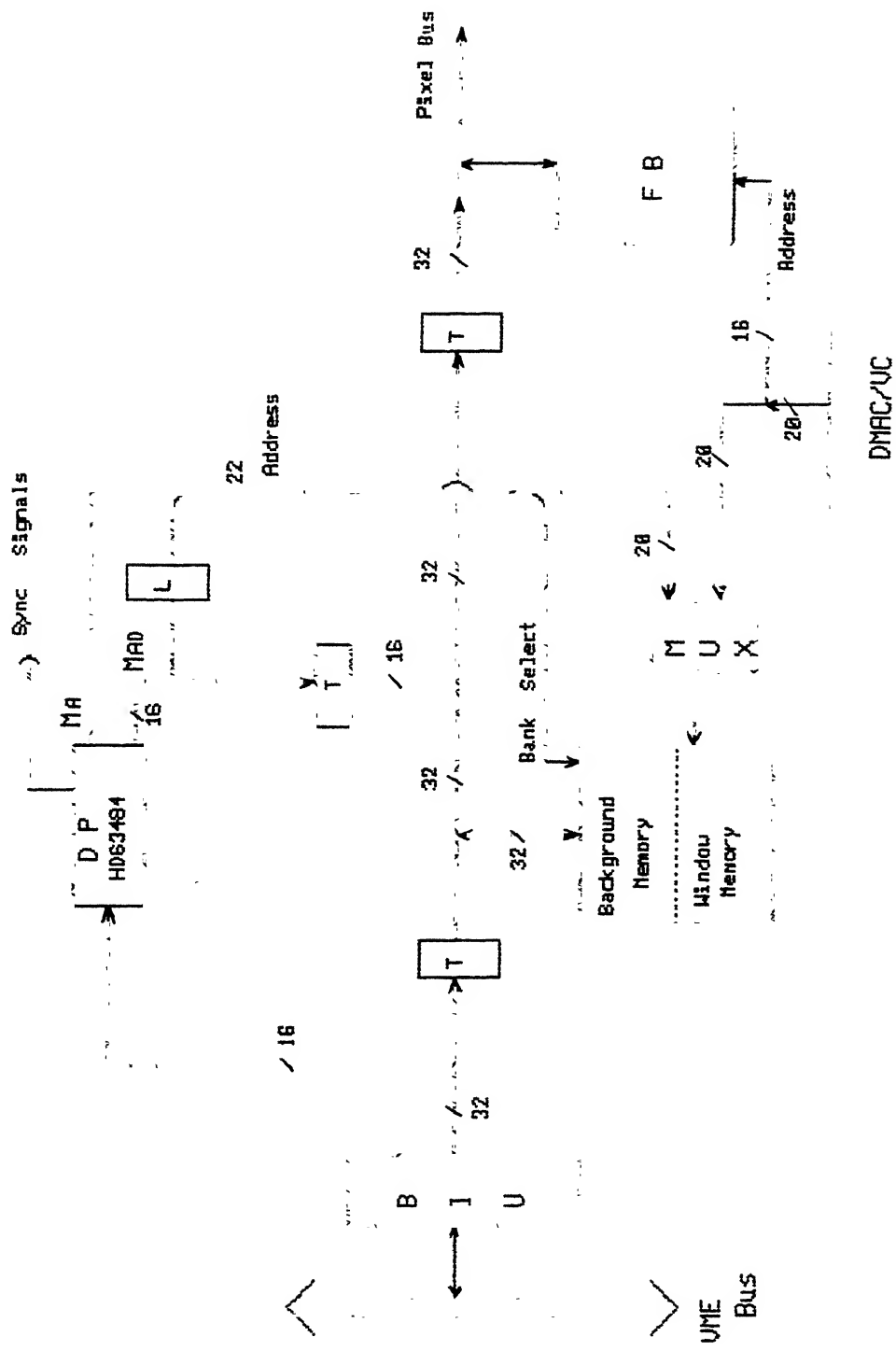


Fig 6.2 Improved FGA

HD63484 which takes care of mundane drawing jobs. ACRTC acting like a Drawing Processor (DP) takes command from host through VME bus and manipulates the copy of FB kept in a additional memory. In this additional memory the windows and background are stored separately. The DMAC, in this implementation, need not request VME bus to update FB but will do a DMA from the additional memory, which is also interfaced to VME bus (like FB interface in FGA), so that in a multiprocessing situations windows can be manipulated by some other processor and then the data is sent to this directly. Thus the additional memory will be accessed for update by 2 different units -- DP and VME bus.

This system configuration envisages high flexibility in terms of image creation in the Background Memory (BM) and the Window Memory (WM). If the screen is windowless then background information can be transferred simultaneously to BM and FB from VME. Since DP (HD63484) does not have capability to transfer 32 bit data, the DP update cannot be done simultaneously in FB and BM. In the case where windows are present, BM/WM can be updated through VME bus or by DP. After this the combined BM and WM data is transferred to FB without hampering the video refresh by utilising SSU cycle.

REFERENCES

- [1] Sherr, Sol, " Video and Digital Electronic Displays : A User's Guide ", John-Wiley and Sons, 1982.
- [2] Leler, William. J., " Human Vision, Anti-aliasing and cheap 4000 line display", ACM Computer Graphics, Vol. 14, No. 3, July 1980, pp 308-313.
- [3] Warnock, John. E., "The Display of characters using gray-level sample array", ACM Computer Graphics, Vol. 14, No.3, July 1980, pp 302-307.
- [4] Felger et. al., " The Realization of a Multiprocessor GKS Architecture ", Parallel Processing for Computer Vision and Display, edited by P. M. Dew et.al., Addison-Wesley Publishing Company, 1989.

ADDITIONAL REFERENCES

- 1. HD 63484 ACRTC Advanced CRT Controller user's manual
- 2. Adding devices to the Horizon III, HCL R&D Madras, 1986.
- 3. VME Bus Specification Manual, Philips, 1985.
- 4. Signetics FAST Data Manual
- 5. B. W. Kernighan and D. M. Ritchie, " The C Programming Language ", Prentice-Hall of India.
- 6. Xenix Development System - C User's Guide, The Santa Cruz Operation, Inc.

APPENDIX 1

This section gives the program listing written to test FGA. The *second* program is the device driver, which has to be compiled as object file and has to be attached to the kernel. Appropriate modifications of switch tables are also required. The *first* program is the user routine, which with the help of device driver, does simple tasks of programming various registers of FGA, and transferring image files to system memory. Both the programs run on HCL Horizon III.

```

/* The following routine interacts directly with the user and programs
   required register. It can also copy the specified image file to the
   system memory area. Due to efficiency considerations it transfers
   file in chunks of 1024 bytes */

```

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/file.h>
#include <sys/ioctl.h>

```

```

/*
This is the list of physical addresses in FGA along with the offsets
which are used in this program

```

```

#define BASE_ADDR      0xffff5c
#define ABS_BREQ        0xffff5c    :+ 0
#define ABS_CRT_ADDREG  0xffff6c    :+ 8
#define ABS_CRT_CNTRL   0xffff6e    :+ 9
#define ABS_AMADDR_LAT  0xffff70    :+ A
#define ABS_WAG         0xffff72    :+ B
#define ABS_STATUS      0xffff74    :+ C
#define ABS_DAG         0xffff76    :+ D
#define ABS_WC1         0xffff78    :+ E
#define ABS_WC2         0xffff7a    :+ F
#define ABS_ZOOM        0xffff7c    :+10
#define ABS-BPP         0xffff7e    :+11
*/

```

```

#define CRT_ADDREG      8
#define CRT_CNTRL       9
#define BUF_WRITE       7
#define BREQ            0
#define POS_INDEX       6
#define DAG_LAT         0xd

```

```

FILE *inpfile;
int fd,c,k,nu,d;
short n,dat,cc,by ;
unsigned int fill,index;
int rval = 0;
short REG_WR=0;
char filename[30];
int val[16]={0x60,0x44,0x50,0x0f,0x20,0x00,0x18,0x19,0x00,0x08,0x00,0x04,
            0x00,0x80,0x00,0x80};

```

```

short over= 0;
/*for each call to device driver T_size Lwords are transferred*/
#define T_size 256
unsigned int locbuf[T_size];

```

```

ain()

```

```

for(;;)
{
    printf("\nFrame buffer Test routine \n");
    printf("OPTIONS:Open, Close, Write, S-6845,Busreq.,D dump,Quit.\n");
    c=getchar();
    k=getchar();      /*dummy to absorb carriage return */

    switch(c)
    {

        case 'O':
            if((fd=open("/dev/fdrv",O_RDWR,0)) <= 0)
            {
                printf("\nUnable to open\n");
                rval = -1;
            }
            else
            {
                printf("\nSucessfully opened..\n");
                rval = 0;
            }
            break;

        case 'C':
            if(close(fd)<0)
            {
                printf("Unable to close \n");
                rval= -1;
            }
            else
            {
                printf("\n Driver closed \n");
                rval=0;
            }
            break;

        case 'W':
            printf("Enter data to be sent :0x  ");
            scanf("%hx",&dat);
            printf("\n..To What Register ( 2 to 9 ) ?.. :");
            scanf("%hd",&REG_WR);
            REG_WR+=8;

            if(ioctl(fd,REG_WR,&dat)<0)
                printf("\nCard write error\n");
            else
            {
                printf("data %hx sent to register %hd \n",dat,REG_WR);
                rval=0;
            }
            k=getchar();      /*dummy to absorb carr. return */
            break;

        case 'S':

```

```

        printf("6845 Initialization ..\n");
        printf(" Writing Pre-programmed values\n");
        for(n=0,rval=0;n<16 && rval!=-1;++n)
        {
            dat=val[n];
            if(ioctl(fd,CRT_ADDREG,&n)<0)
            {
                printf("\n6845 Addr_reg Write error\n");
                rval= -1;
            }
            if(ioctl(fd,CRT_CNTRL,&dat)<0)
            {
                printf("\n6845 Cntrl_reg write error\n");
                rval= -1;
            }
        }
    }
    break;

    case 'B':
        if(ioctl(fd,BREQ,&dat)<0)
            printf("\nBus Request card write error\n");
        else
        {
            printf("..Device Wants Bus flag is set..\n");
            rval=0;
        }
    break;

    case 'D':
        /*dumps 1Kb of the specified file in the local buffer
        then calls device driver to transfer locbuf */

        printf("From what file ?\n");
        scanf ("%s",filename);
        if((infile=fopen(filename,"r"))!=NULL)
        {
            printf("file opened Wait...Dumping\n");
            while(over!=1 && rval!= -1)
            {
                index=0;
                while (over!=1 && index<T_size)
                {
                    if(fscanf(infile,"%x",&fill)==EOF)
                        over = 1;
                    else
                    {
                        locbuf[index]=fill;
                        ++index;
                    }
                }
                if(ioctl(fd,BUF_WRITE,locbuf)<0)
                {
                    printf("Buffer Write error\n");

```

```

        rval = -1;
    }
}
if(rval==0)
    printf("\n128Kb dumped\n");
if(ioctl(fd,POS_INDEX,&dat)<0)
    printf("Unable to reinitialize index\n")
else
    printf("index repositioned\n");
over=0;
}
else
    printf("Unable to open dump file\n");

break;

case 'Q':
    printf("Quitting...\n");
    return(0);

break;

}
}
}

```

```

/*This is the device driver written for FGA - routines like svtop are u
to Horizon workstation, similar routines may also be available in
machines although in a different name */

#define MAXBUFLen 255
#define MAXNAMLEN 255
#define NGROUPS 8
#define NSIG 32
#define NOFILE 20

/*frbuf is the memory area in system memory from where the FGA will read
image data and fill in its frame buffer, frbuf is defined to be two
128 Kbytes (32k LW) so as to align the memory area seen by the FGA
a 128k page boundary ... half the memory is unused in this process

#define BUF_LEN 0x40000 /* 32k LW */
#define BUF_WRITE 7 /* assigned a number to indicate to
device driver that frbuf is to be fil
#define POS_INDEX 6 /* the index to frbuf is to be initialized*/
#define DAG_LAT 0xd
#include <sys/types.h>
#include <sys/errno.h>
#include <sys/buf.h>

struct direct {
    u_long d_ino;
    u_short d_reclen;
    u_short d_namlen;
};
struct sigstack {
    char *ss_sp;
    int ss_onstack;
};

#include <sys/user.h>
extern struct user u ;

/* lowest address of FGA is INIT - all other addresses are derived by ad
the parameter passed from user routine */

u_short *FR_ADDR = (u_short *) 0xffff5c ;
u_short *INIT = (u_short *) 0xffff5c ;
u_short *ADDR_LAT = (u_short *) 0xffff70 ; /* Page Address Register add
char frbuf[BUF_LEN];
char *usad ;
u_int bufphy;
u_short bnksel=0; /* A_17 bit to select one of 128Kb banks -
at present left unconnected in hardware */
char *index;
u_int staphy,tocon;
u_int con = 0;
u_short n,clr;

frioctl(dev,cmd,arg,mode)

```

```

dev_t dev ;
int cmd,mode;
caddr_t *arg;
{
    u_short wrtw ;
    usad = (char *)u.u_arg[2];

switch (cmd)
{
    case PDS_INDEX:
        index=fdbuf+(staphy-bufphy) ; /* initialize the index
                                         point to start of fdbuf */
        return(0);
    break;

    case BUF_WRITE:
        if(index<((fdbuf+BUF_LEN))) /* check upper bound */
        {
            if(copyin(usad,index,1024)<0)
            {
                cprintf ("buffer copy_in error \n");
                return(-1);
            }

            index+=1024;
        }
        else
            return(-1);

    break;

default:
        /* for programming all other registers */
        if(copyin(usad,(char*)&wrtw,2)<0)
        {
            cprintf ("copy_in error \n");
            return(-1);
        }

        if((cmd==DAG_LAT) && (bnksel)) /*if DAG write then A_17
                                         wrtw!=0x1000;      /* is written */

        FR_ADDR+=cmd;
        cprintf ("WRITING 0x%x at 0x%x \n",wrtw,FR_ADDR);
        *FR_ADDR = wrtw ;
        FR_ADDR = INIT;
        return(0);

    break;
}

return(0);

```

}

```
open(dev)
v_t dev;
```

```
bufphy = svtop(frbuf);
cprintf("buffer allocated phys address:0x%x\n",bufphy);
cprintf("buffer allocated virtual address:0x%x\n",frbuf);
```

```
/*following line finds the paged buffer origin;
    >>2 if frbuf is defined as a int*/
```

```
staphy=(bufphy+ 0x1ffff) & ~0x1ffff ;
cprintf("starting physical address is 0x%x\n",staphy);
```

```
index=frbuf+(staphy-bufphy); /* apart from subtraction division
    is required if buf is defined as
cprintf("filling will start from (V.addr) 0x%x\n",index);
```

```
/* the following lines convert the physical address
of the buffer to the form that can be put into
the address latch directly */
```

```
con=0;
tocon= ( staphy & 0xff0000) >> 16;
con!=(tocon & 01);
for (n=7 ;n>0 ;--n)
{
    tocon>>=1;
    con<<= 1;
    con!=(tocon & 01);
}
```

```
cprintf("shifted and reversed hig. order addr. bits:0x%x\n",con)
bnksel=con & 0x40; /*bnksel is the A_17 bit of address*/
```

```
con =((con & 0x003f) << 8)!0x00f9; /*A_18 to A_23 is retained
and address modifier is set */
```

```
/*The control word DRed is 0x0079 in VME bus systems capable
of doing 32 bit data transfers ;0x00f9 for 16 bit systems
like Horizon */
```

```
*ADDR_LAT = con ;
cprintf ("Page Address Register is set(for write) to 0x%x\n",con)
```

```
if(bnksel)
    cprintf("A_17 bit is set\n");
```

```
else
    cprintf("A_17 is zero \n");
cprintf (" FR : opened \n ");
cprintf ("Clearing all registers.. ");
for(clr=1;clr<8;++clr)
    *(ADDR_LAT+clr)=0;
```

```
        cprintf("done\n");  
        return(0);
```

```
    }
```

```
frclose(dev)
```

```
dev_t dev;
```

```
{
```

```
    cprintf ("FR : closed \n " );
```

```
    return(0);
```

```
}
```

TH 006.6 Date Slip A109949
G1548 This book is to be returned on the
date last stamped.

EE-1990-M-GAN-FLE